VIENT LISTINGS SUPPLEMENT LISTINGS SUPPLEMENT NGS SUPPLEMENT LISTINGS SUPPLEMENT LISTINGS SUPPLE LISTINGS SUPPLEMENT LISTINGS SUPPLEMENT LISTINGS UPPLEMENT LISTING SUPPLEMENT LISTINGS SUPPLEMENT LISTINGS

WENT LISTINGS SUPPLEMENT LISTINGS

LI SUPPLEMENT LISTINGS SUPPLEMENT SUPPLEMENT LISTINGS SUPPLEMENT SUP ISTINGS SUPPLEMENT LISTINGS SUPPLEMENT LISTINGS SU GS SUPPLEMENT LISTINGS SUPPLEMENT LISTINGS SUPPLEMENT PPLEMENT LISTINGS SUPPLEMENT LISTINGS SUPPLEMENT, MENT LISTINGS SUPPLEMENT LISTINGS SUPPLEMENT LISTINGS LISTINGS SUPPLEMENT LISTINGS SUPPLEMENT LISTINGS S INGS SUPPLEMENT LISTINGS SUPPLEMENT LISTINGS SUPPLEMENT SUPPLEMENT LISTINGS SUPPLEMENT LISTINGS SUPPLEMENT EMENT LISTINGS SUPPLEMENT LISTINGS SUPPLEMENT LIST T LISTINGS SUPPLEMENT LISTINGS SUPPLEMENT SUPPLEMENT LISTINGS SUPPLEMENT LISTINGS SUPPLEMENT EMENT LISTINGS SUPPLEMENT LISTINGS SUPPLEMENT LIST T LISTINGS SUPPLEMENT LISTINGS SUPPLEMENT LISTINGS

Six great reasons to join BIX today

Over 140 microcomputer-related conferences:

Join only those subjects that interest you and change selections at any time. Take part when it's convenient for you. Share information, opinions and ideas in focused discussions with other BIX users who share your interests. Easy commands and conference digests help you quickly locate important information.

Monthly conference specials:

BIX specials connect you with invited experts in leading-edge topics—CD-ROM, MIDI, OS-9 and more. They're all part of your BIX membership.

Microbutes dailu:

Get up-to-the-minute industry news and new product information by joining Microbytes Daily and What's New Hardware and Software.

• Public domain software:

Yours for the downloading, including programs from BYTE articles and a growing library of PD listings.

• Electronic mail:

Exchange private messages with BYTE editors and authors and other BIX users.

Vendor support:

A growing number of microcomputer manufacturers use BIX to answer your questions about their products and how to use them for peak performance.

What BIX Costs. . How You Pay

ONE-TIME REGISTRATION FEE: \$25

Hourly Charges: (Your Time of Access)	Off-Peak 6PM-7AM Weekdays Plus Weekends & Holidays	Peak 7AM-6PM Weekdays
BIX Tymnet*	\$9 \$2	\$12 \$6
TOTAL	\$11/hr.	\$18/hr.**

^{*} Continental U.S. BIX is accessible via Tymnet from throughout the U.S. at charges much less than regular long distance. Call the BIX helpline number listed below for the Tunnet number near you or Tymnet at 1-800-336-0149

BIX and Tymnet charges billed by Visa or Mastercard only.

BIX HELPLINE

(8:30 AM-11:30 PM Eastern Weekdays)

U.S. (except NH)—1-800-227-BYTE Elsewhere (603) 924-7681



BIX User's Manual and Subscriber Agreement as Soon as We've Processed Your Registration. JOIN THE EXCITING WORLD OF BIX TODAY!

IOIN BIX RIGHT NOW:

Set your computer's telecommunications program for full duplex, 8-bit characters, even parity, 1 stop bit OR 7-bit characters, even parity, 1 stop using 300 or 1200 baud.

Call your local Tymnet number and respond as follows:

Tymnet Prompt	You Enter
Garble or "terminal identifier" login: password: mhis login: BIX Logo—Name:	a byteneti < CR > mgh < CR > bix < CR > new < CR >
Diri bogo iraine.	Her Che

After you register on-line, you're immediately taken to the BIX learn conference and can start using the system right away.

FOREIGN ACCESS:

To access BIX from foreign countries, you must have an account with your local Postal Telephone & Telegraph (PTT) company. From your PTT enter 310600157878. Then enter bix <CR> and new <CR> at the prompts. Call or write us for PTT contact information.



ONE PHOENIX MILL LANE PETERBOROUGH, NH 03458 (603) 924-9281

^{**} User is billed for time on system (i.e., ½ Hr. Off-Peak w/Tymnet = \$5.50 charge.)

TABLE OF CONTENTS

July	SHOTIGS SHOW BALLOCKA	5
Six : Alch Cox	Anne Piether Lent, What's New. Stan Macklowski, What's New. Bett of	
September	Mark Detroy, record of the Artist Mark Hazarda	89

WELCOME TO BYTE'S QUARTERLY LISTINGS SUPPLEMENT

The BYTE Listings Supplement is produced quarterly as a means of providing interested readers with a printed, source code version of those programs referenced in BYTE articles. It provides a far more extensive look into the techniques of coding and the potentialities of microcomputers than we have space for in each month's BYTE.

Programs contained in this Supplement are referenced by the month the article appeared, the page on which their supporting article begins, and the name of the author who wrote the article.

For those who prefer programs already in electronic format, we have a companion service called Listings on Disk. If you have a modem, listings may be downloaded from the BYTEnet bulletin board and, if you are a member of BIX, the "Listings" area also contains programs referenced in BYTE.

If you live outside of the U.S., we've included the names, addresses and telephone numbers of bulletin boards that get program code from us. You'll find the directory just inside the back cover of this Supplement.

The bulletin boards are updated monthly. Several countries have enough boards that the telephone charges for most callers should be the minimum possible.

HUI-

EXECUTIVE EDITOR, BYTE Frederic S. Langa

ASSISTANT MANAGING EDITOR Glenn Hartwig

CONSULTING EDITORS Steve Ciarcia Jerry Pournelle Ezra Shapiro

SENIOR TECHNICAL EDITORS Cathryn Baskin, Reviews G. Michael Vose, Themes Gregg Williams, Features

TECHNICAL EDITORS Dennis Allen Curtis Franklin Jr. Richard Grehan Ken Sheldon George A. Stewart Jane Morrill Tazelaar Tom Thompson Charles D. Weston **Eva White** Stanley Wszola

ASSOCIATE TECHNICAL EDITORS Martha Hicks

COPY EDITORS Lauren Stickler, Copy Administrator Judy Connors-Tenney Jeff Edmonds Nancy Hayes Cathy Kingery Margaret A. Richard Warren Williamson

ASSISTANTS Peggy Dunham, Office Manager Ryan McCombs June N. Sheldon

NEWS AND TECHNOLOGY

Gene Smarte, Bureau Chief, Costa Mesa Jonathan Erickson, Senior Technical Editor, San Francisco Rich Malloy, Senior Technical Editor, New York Nicholas Baran, Associate Technical Editor, San Francisco

ASSOCIATE NEWS EDITORS Dennis Barker, Microbytes Anne Fischer Lent, What's New Stan Miastkowski, What's New, Best of BIX

CONTRIBUTING EDITORS Jonathan Amsterdam, programming projects Mark Dahmke, video, operating systems Mark Haas, at large Rik Jadrnicek, CAD, graphics, spreadsheets Robert T. Kurosaka, mathematical recreations Alastair J.W. Mayer, software Alan R. Miller, languages and engineering Dick Pountain, U.K Roger Powell, computers and music Phillip Robinson, semiconductors Jon Shiell, high-performance systems

Nancy Rice, Art Director Joseph A. Gallagher, Assistant Art Director Jan Muller, Art Assistant Alan Easton, Drafting

PRODUCTION
David R. Anderson, Production Director Denise Chartrand Michael J. Lonsky Virginia Reardon

TYPOGRAPHY Sherry McCarthy, Chief Typographer Selinda Chiquoine Donna Sweeney

EXECUTIVE EDITOR, BIX George Bond

SENIOR EDITOR David Betz

ASSOCIATE EDITORS Tony Lockwood Donna Osgood, San Francisco

MICROBYTES DAILY Dennis Barker, Coordinator, Peterborough Gene Smarte, Bureau Chief, Costa Mesa Nicholas Baran, San Francisco Rick Cook, Phoenix Hick Cook, Phoenix Jonathan Erickson, San Francisco Martha Hicks, Peterborough Anne Fischer Lent, Peterborough Larry Loeb, Wallingford, CT Rich Malloy, New York Brock N. Meeks, La Mesa, CA Jeff Merron, Peterborough Stan Miastkowski, Peterborough Lynne Nadeau, Peterborough Wayne Rash, Washington, DC

GROUP MODERATORS David Allen, Applications Frank Boosman, Artificial Intelligence Leroy Casterline, Other Marc Greenfield, Programming Languages Jim Howard, Graphics
Gary Kendall, Operating Systems
Steve Krenek, Computers
Brock N. Meeks, Telecommunications
Barry Nance, New Technology Donald Osgood, Computers Sue Rosenberg, Other Jon Swanson, Chips

BUSINESS AND MARKETING Doug Webster, Director (603-924-9027) Patricia Bausum, Secretary Denise A. Greene, Customer Service Brian Warnock, Customer Service Tammy Burgess, Customer Credit and Billing

TECHNOLOGY Clayton Lisle, Director Business Systems Technology, MHIS Bill Garrison, Business Systems Analyst Jack Reilly, Business Systems Analyst

Officers of McGraw-Hill Information Systems Company: President: Richard B. Miller. Executive Vice Presidents: Frederick P. Jannott, Construction Information Group; Russell C. White, Computers and Communications Information Group; J. Thomas Ryan, Marketing and International. Senior Vice Presidents: Francis A. Shinal, Controller; Robert C. Violette, Manufacturing and Technology. Senior Vice Presidents and Publishers: Laurence Altman, Electronics; Harry L. Brown, BYTE; David J. McGrath, Construction Publications. Group Vice President: Peter B. McCuen, Communications Information. Vice President: Fred O. Jensen, Planning and Developmen

Officers of McGraw-Hill, Inc.: Harold W. McGraw, Jr., Chairman; Joseph L. Dionne, President and Chief Executive Officer; Robert N. Landes, Executive Vice President and Secretary; Walter D. Serwatka, Executive Vice President and Chief Financial Officer; Shel F. Asen, Senior Vice President, Manufacturing; Robert J. Bahash, Senior Vice President, Finance and Manufacturing; Ralph R. Schulz, Senior Vice President, Editorial; George R. Elsinger, Vice President, Circulation; Ralph J. Webb, Vice President and Treasurer.

BYTE. , and The Small Systems Journal are registered trademarks of McGraw-Hill Inc.

EDITORIAL AND BUSINESS OFFICE: One Phoenix Mill Lane, Peterborough, New Hampshire 03458, (603) 924-9281.

West Coast Offices: 425 Battery St., San Francisco, CA 94111, (415) 954-9718; 3001 Red Hill Ave., Building #1, Suite 222, Costa Mesa, CA 92626, (714) 557-6292. New York Editorial Office: 1221 Avenue of the Americas, New York, NY 10020, (212) 512-3175.

BYTEnet: (617) 861-9764 (set modem at 8-1-N or 7-1-E; 300 or 1200 baud).

BYTE (ISSN 0360-5280) is published monthly with additional issues in June and October by McGraw-Hill Inc. Founder: James H. McGraw (1860–1948). Executive, editorial, circulation, and advertising offices: One Phoenix Mill Lane, Peterborough, NH 03458, phone (603) 924-9281. Office hours: Monday through Thurs-Phoenis Mill Lane, Peterborough, NH 03458, phone (603) 924-9281. Office hours: Monday through Thursday 8:30 AM-4:30 PM, Friday 8:30 AM-1:00 PM, Eastern Time. Address subscriptions to BYTE Subscriptions, P.O. Box 590, Martinsville, NJ 08836. Postmaster: send address changes, USPS Form 3579, undeliverable copies, and fulfillment questions to BYTE Subscriptions, P.O. Box 596, Martinsville, NJ 08836. Second-class postage paid at Peterborough, NH 03458 and additional mailing offices. Postage paid at Winnipeg, Manitoba. Registration number 9321. Subscriptions are \$22 for one year, \$46 for two years, and \$58 for three years in the U.S. and its possessions. In Canada and Mexico, \$25 for one year, 445 for two years, \$65 for three years. \$69 for one year are delivery to Europe. 31,000 yen for one year art delivery to Japan, \$37 surface delivery elsewhere. Air delivery to selected areas at additional rates upon request. Single copy price is \$3.50 in the U.S. and its possessions, \$4.25 in Canada and Mexico, \$4.50 in Europe, and \$56 elsewhere. Foreign subscriptions and sales should be remitted in U.S. funds farms on \$4.50 in Europe, and \$5 elsewhere. Foreign subscriptions and sales should be remitted in U.S. funds drawn or a U.S. bank. Please allow six to eight weeks for delivery of first issue. Printed in the United States of America.

a U.S. bank. Please allow six to eight weeks for delivery of first issue. Printed in the United States of America. Address editorial correspondence to: Editor, BYTE, One Phoenix Mill Lane, Peterborough, NH 03458. Unacceptable manuscripts will be returned if accompanied by sufficient postage. Not responsible for lost manuscripts or photos. Opinions expressed by the authors are not necessarily those of BYTE.

Copyright © 1987 by McGraw-Hill Inc. All rights reserved. Trademark registered in the United States Patent and Trademark Office. Where necessary, permission is granted by the copyright owner for libraries and others registered with the Copyright Clearance Center (CCC) to photocopy any article herein for the flat fee of \$1.50 per copy of the article or any part thereof. Correspondence and payment should be article or only part thereof. Correspondence and payment should be and irectly at the CCC, 29 Congress St., Salem, MA 01970. Specify ISSN 0360-5280/83. \$1.50 Copying done for other than personal or internal reference use without the permission of McGraw-Hill Inc. is prohibited. Requests for special permission or bulk orders should be addressed to the publisher. BYTE is available in microform from University Microfilms International, 300 North Zeeb Rd., Dept. PR, Ann Arbor, MI 48106 or 18 Bedford Row, Dept. PR, London WCIR 4EJ, England. Subscription questions or problems should be addessed to: BYTE Subscriber Service, P.O. Box 328, Hancock, NH 03449.

INDEX

July		QLACE	CRV40 CRV42
BENCH	ADA5	QMDLQUIN QMOORE	CRV42
DHRY	C13	QPENTGRE	CRV38
FIB	C23 C26	QRULES	TXT50
FILEIO FLOAT	C23	QSNOFLK QSRPNSK	CRV38 CRV38
LISTING	TXT35	QSRPNSK2	CRV42
LISTING2	TXT33	SAVAGE	C82
SAVAGE	C26	SIEVE	C80
SIEVE	C24	SORT	C81
SORT	C24 PAS31	START80 STARTUP	ARI79 C76
WHET	C27	SZPAK	BNL46
	0	SZPAK	LST44
		TOKENS80	ARI77
August			
		Septembe	r
CHAOSBEN	BAS87		
DONE DRAGON	C84 BAS49	ADD—PATT	C 124
FIB	C84	ALSPRO	TST89
FILEIO	C85	ASYNCH	C 126
FLOAT	C83	AWAIT	C 126
HELP80	ARI67	BAM	BAS 116
INDEX	PAS53	BAM	PAS99
INTERFAC	C78 ARI76	BAM CLEARBAM	C 132 C 133
MEMORY MEMORY	BAS78	EXT	H 133
MON80	ARI74	GET-MTX	C 131
OPS8085	ARI55	KAREX1	BAS 140
PARSE80	ARI72	KAREX2	BAS 138
PREDS80	ARI69	KAREX3	BAS97
QAROHEAD	CRV39	PAT	1 132 BAM 125
QBRICK QBRKINT	CRV40 CRV40	README SIEVE386	ASM 127
OCHRSTRE	CRV43	SIEVE86	ASM 134
QDEKNGCH	CRV37	SORT	MOD 124
QDRAGON	BAS51	SORT	DEF 125
QDRGBDRY	CRV41	SORTELEM	MOD 121
QDRGN	CRV44	SORTELEM	DEF 123
QDRGNCRD ODRGNINT	CRV43 CRV37	SORTTEST TURBFLOP	MOD 122 PRO 138
OGOSPER	CRV44	XFACE	INC 114
QHILBRT	CRV37	ATACE	

July

BENCH. ADA Contributed by Namir Clement Shammas. Listings accompany the review of four Ada compilers: Alsys Ada, Artek Ada, AdaVantage, and JANUS Ada, "Ada Moves to Micros," July 1987, page 239.

```
Listing 1: Source code for Ada Sieve benchmark program.
with TEXT_IO;
use TEXT_IO;
-- package INTIO is new INTEGER_IO(INTEGER);
PROCEDURE MTSIE10 is
SIZE : constant INTEGER := 7000;
TYPE Flag_Array is array(0..SIZE) of BOOLEAN;
PRIME, K, COUNT : INTEGER;
FLAGS : Flag_Array;
BEGIN
  PUT_LINE("START TEN ITERATIONS");
  FOR ITER IN 1..10 LOOP
    COUNT := 0;
    FOR I IN O. . SIZE LOOP
      FLAGS(I) := TRUE;
     END LOOP;
     FOR I IN O. . SIZE LOOP
       IF FLAGS(I) THEN
         PRIME := I + I + 3;
         K := I + PRIME;
         WHILE K <= SIZE LOOP
           FLAGS(K) := FALSE;
           K := K + PRIME;
         END LOOP;
         COUNT := COUNT + 1;
       END IF;
     END LOOP;
 END LOOP;
  PUT(INTEGER'IMAGE(COUNT));
  PUT_LINE(" PRIMES");
END MTSIE10;
Listing 2: Source code for Ada integer Sort benchmark program.
with TEXT_IO;
use TEXT_IO;
Procedure MTSort2 is
-- Program will test the speed of sorting an integer array.
 -- The program will create an array sorted from smaller to larger
-- integers, then sort them in the reverse order.
 -- The array is reverse-sorted ten times.
```

```
package INTIO is new INTEGER_IO(INTEGER);
SIZE : constant := 1000;
TYPE NUMBERS is ARRAY(1..SIZE) OF INTEGER;
InOrder, AscendingOrder: BOOLEAN;
Offset, Temporary : INTEGER;
Ch : CHARACTER;
A : NUMBERS;
PROCEDURE InitializeArray is
-- Procedure to initialize array
BEGIN
  PUT_LINE("Initializing integer array");
  FOR I IN 1..SIZE LOOP
    A(I) := I;
  END LOOP;
END InitializeArray;
PROCEDURE ShellSort is
-- Procedure to perform a Shell-Metzner sorting
I : INTEGER;
 PROCEDURE SwapThem(I, J: in INTEGER) is
  -- Local procedure to swap elements A(I) and A(J)
  InOrder := FALSE;
  Temporary := A(I);
  A(I) := A(J);
  A(J) := Temporary;
 END SwapThem;
 BEGIN
  -- Toggle "AscendingOrder" flag status
     AscendingOrder := NOT AscendingOrder;
    Offset := SIZE;
    WHILE Offset > 1 LOOP
      Offset := Offset/2;
       LOOP
         InOrder := TRUE;
         FOR J IN 1.. (SIZE - Offset) LOOP
           I := J + Offset;
           IF AscendingOrder
             THEN IF A(I) < A(J) THEN SwapThem(I,J); END IF;
             ELSE IF A(I) > A(J) THEN SwapThem(I,J); END IF;
           END IF; -- AscendingOrder
         END LOOP;
         IF InOrder THEN EXIT; END IF;
      END LOOP;
    END LOOP;
 END ShellSort;
 PROCEDURE DisplayArray is
 -- Display array members
 BEGIN
   FOR I IN 1..SIZE LOOP
     INTIO.PUT(A(I),3);
     PUT(" ");
   END LOOP;
   NEW_LINE;
 END DisplayArray;
 BEGIN -- Main
   InitializeArray;
   AscendingOrder := TRUE;
   PUT("Beginning to sort press <cr> "); GET(Ch); NEW_LINE;
   FOR Iter IN 1..10 LOOP
   PUT(".");
    ShellSort;
  END LOOP;
   PUT_LINE("Finished sorting!");
  DisplayArray;
END MTSort2;
```

```
Listing 3: Source code for Ada basic Floating benchmark program.
WITH TEXT_IO; USE TEXT_IO;
PROCEDURE MTFLOAT is
PACKAGE RealInOut is new FLOAT_IO(FLOAT);
USE RealInOut;
NR : CONSTANT INTEGER := 5000;
A, B, C : FLOAT;
BEGIN
  A := 2.71828;
  B:= 3.1459;
  C := 1.0;
  FOR I IN 1..NR LOOP
   C := C * A;
    C := C * B;
    C := C / A;
   C := C/B;
  END LOOP;
  PUT("DONE");
  NEW_LINE;
  PUT("ERROR = ");
  PUT((C-1.0));
  NEW_LINE;
END MTFLOAT;
Listing 4: Source code for Ada matrix-inversion Floating benchmark program.
with TEXT_IO;
use TEXT_IO;
Procedure MTINVERT is
-- Program to test speed of floating-point matrix inversion.
-- The program will form a matrix with 1s in every member,
-- except the diagonals which will have values of 2.
 package RealInOut is new FLOAT_IO(FLOAT);
 MAX : constant := 20;
 TYPE MATRIX is ARRAY (1..MAX, 1..MAX) OF FLOAT;
 J, K, L: INTEGER;
 DET, PIVOT, TEMPO: FLOAT;
 A: MATRIX;
Procedure Invert is
-- Creating test matrix
 FOR J IN 1. . MAX LOOP
  FOR K IN 1. . MAX LOOP
   A(J, K) := 1.0;
  END LOOP:
  A(J, J) := 2.0;
 END LOOP:
 PUT_LINE("Starting matrix invertion");
 DET := 1.0;
```

```
FOR J IN 1. . MAX LOOP
  PIVOT := A(J, J);
  DET := DET*PIVOT;
  A(J, J) := 1.0;
  FOR K IN 1. . MAX LOOP
   A(J, K) := A(J, K) / PIVOT;
  END LOOP;
  FOR K IN 1. . MAX LOOP
    IF K /= J THEN
     TEMPO := A(K, J);
     A(K, J) := 0.0;
     FOR L IN 1. . MAX LOOP
      A(K, L) := A(K, L) - A(J, L) * TEMPO;
     END LOOP;
   END IF;
  END LOOP;
 END LOOP;
END Invert;
BEGIN
 NEW_LINE(2);
 Invert;
 PUT("Determinant = ");
 RealInOut.PUT(DET, 14, 10);
 NEW_LINE(2);
END MTINVERT;
Listing 5: Source code for Ada Math Functions benchmark program.
-- use Janus/Ada libraries
WITH TEXT_IO; WITH SMATHLIB;
USE TEXT_IO; USE SMATHLIB;
PROCEDURE MTMath is
-- Program tests the speed of math function.
-- Each function is timed separately.
-- Functions are shown in the import list.
pi, angle, result, argument: FLOAT;
dummy: CHARACTER;
BEGIN
PUT_LINE("START SQUARE ROOT TEST");
PUT("PRESS < CR > TO START");
GET(dummy); New_Line;
 FOR i in 1..10 LOOP
 PUT(".");
  argument := 0.0;
 WHILE argument <= 1000.0 LOOP
  result := Sqrt(argument);
  argument := argument + 1.0;
 END LOOP;
END LOOP;
New_Line; PUT("END OF SQUARE ROOT TEST"); New_Line;
```

```
PUT("START LOG TEST");
New_Line;
PUT("PRESS < CR > TO START");
GET(dummy); New_Line;
FOR i in 1..10 LOOP
 PUT(".");
 argument := 0.1;
 WHILE argument <= 1000.1 LOOP
  result := Log(argument);
  argument := argument + 1.0;
 END LOOP;
END LOOP;
New_Line; PUT("END OF LOG TEST"); New_Line;
PUT("START EXPONENTIAL TEST");
New_Line;
PUT("PRESS < CR> TO START");
GET(dummy); New_Line;
FOR i in 1..10 LOOP
 PUT(".");
 argument := 0.1;
 WHILE argument <= 10.0 LOOP
  result := exp(argument);
  argument := argument + 0.01;
 END LOOP;
END LOOP;
New_Line; PUT("END OF EXPONENTIAL TEST"); New_Line;
PUT("START ARCTANGENT TEST");
New_Line;
PUT("PRESS < CR > TO START");
GET(dummy); New_Line;
FOR i in 1..10 LOOP
 PUT(".");
 argument := 0.1;
 WHILE argument <= 10.0 LOOP
  angle := arctan(argument);
  argument := argument + 0.01;
 END LOOP;
END LOOP;
New_Line; PUT("END OF ARCTANGENT TEST"); New_Line;
pi := 355.0/113.0;
PUT("START SINE TEST");
New_Line;
PUT("PRESS < CR > TO START");
GET(dummy); New_Line;
FOR i in 1..10 LOOP
 PUT(".");
 angle := 0.0;
 WHILE angle <= 2.0 * pi LOOP
  result := sin(angle);
  angle := angle + pi/360.0;
 END LOOP;
END LOOP;
New_Line; PUT("END OF SINE TEST"); New_Line;
New_Line;
PUT("DONE"); New_Line; New_Line;
Listing 6: Source code for Ada Recursion benchmark program.
with TEXT_IO;
use TEXT_IO;
Procedure MTQSort is
```

```
-- The test uses QuickSort to measure recursion speed.
-- An ordered array is created by the program and is
-- reverse-sorted. The process is performed "MAXITER"
-- number of times.
package Int_IO is new INTEGER_IO(INTEGER);
SIZE : constant := 1000;
MAXITER : constant := 10;
WantToListArray : constant BOOLEAN := FALSE; -- Flag used for debugging
TYPE Numbers is ARRAY(1..SIZE) OF INTEGER;
A: Numbers;
PROCEDURE InitializeArray is
-- Procedure to initialize array
  FOR I in 1..SIZE LOOP
    A(I) := SIZE - I + 1;
  END LOOP;
  NEW_LINE(3);
END InitializeArray;
PROCEDURE QuickSort is
-- Procedure to perform a QuickSort
PROCEDURE Sort(Left, Right: INTEGER) is
i, j : INTEGER;
Data1, Data2 : INTEGER;
BEGIN
  i := Left; j := Right;
  Data1 := A((Left + Right)/2);
    WHILE A(i) < Data1 LOOP i := i + 1; END LOOP;
    WHILE Data1 < A(j) LOOP j := j - 1; END LOOP;
    IF i <= j THEN
      Data2 := A(i); A(i) := A(j); A(j) := Data2;
      i := i + 1;
      j := j - 1;
    END IF;
    IF i > j THEN EXIT; END IF;
  END LOOP;
  IF Left < j THEN Sort(Left, j); END IF;</pre>
  IF i < Right THEN Sort(i, Right); END IF;</pre>
END Sort;
BEGIN
  Sort(1,SIZE);
END QuickSort;
PROCEDURE DisplayArray is
-- Display array members
BEGIN
  FOR I in 1..SIZE LOOP
    Int_IO.PUT(A(I),4);
    PUT(" ");
  END LOOP;
  NEW_LINE;
END DisplayArray;
BEGIN -- Main
  FOR Iter in 1..MAXITER LOOP
   InitializeArray;
   PUT(".");
   QuickSort;
  END LOOP;
  NEW_LINE;
  PUT_LINE("Finished sorting!");
  IF WantToListArray THEN DisplayArray; END IF;
END MTQSort;
```

Listing 7: Source code for Ada Dynamic Allocation benchmark program.

```
with TEXT_IO;
use TEXT_IO;
PROCEDURE MTPtr is
-- Program to measure the speed of:
-- 1) Allocating dynamic binary-tree structure
-- 2) Searching through the binary tree
SIZE : constant INTEGER := 1000;
MainLoopCount : constant INTEGER := 200;
TYPE Node;
TYPE Ptr is access Node;
TYPE Node is record
       Value : INTEGER;
       Left, Right: Ptr;
      end record;
TYPE NumbersArray is ARRAY (1...SIZE) OF INTEGER;
Numbers : NumbersArray;
TreeRoot : Ptr;
dummy : CHARACTER;
PROCEDURE Create is
J : INTEGER := 1;
BEGIN
  WHILE J <= SIZE LOOP
    IF (J>= 1) AND (J < 251) THEN
     Numbers(J) := J;
    ELSIF (J > 250) AND (J < 501) THEN
    Numbers(J) := SIZE - J;
    ELSIF (J > 500) AND (J < 750) THEN
    Numbers(J) := J;
     Numbers(J) := SIZE - J;
    END IF;
    J := J + 1;
    PUT(INTEGER'IMAGE(J) & " ");
  END LOOP;
  new_line;
END Create;
PROCEDURE Insert(Root: in out Ptr; Item: INTEGER) is
-- Insert element in binary tree
BEGIN
  IF Root = null THEN
    Root := new Node;
    Root. Value := Item;
    Root.Left := null;
    Root.Right := null;
  ELSE
    IF Item < Root. Value THEN Insert(Root.Left, Item);
                        ELSE Insert(Root.Right, Item);
    END IF;
  END IF;
END Insert;
PROCEDURE Search (Root: in out Ptr; Target: INTEGER) is
-- Recursive procedure to search for Target value
```

```
BEGIN
   IF not (Root = null) THEN
     IF not (Target = Root. Value) THEN
       IF Target < Root. Value THEN
         Root := Root.Left; Search(Root, Target);
       ELSE
         Root := Root.Right;
         Search(Root, Target);
       END IF;
     END IF:
  END IF:
END Search:
BEGIN -- MAIN
  Create;
  PUT_LINE("Created array");
   -- Building the binary tree
  PUT("Press < CR> to time tree creation");
  GET(dummy); NEW_LINE;
  TreeRoot := null;
   FOR I IN 1..SIZE LOOP
   Insert(TreeRoot, Numbers(I));
   END LOOP;
  NEW_LINE;
  PUT_LINE("Created Tree");
  PUT("Press <CR> to time tree search");
  GET(dummy); NEW_LINE;
  FOR Iter IN 1.. MainLoopCount LOOP
    FOR I IN reverse 1..SIZE LOOP
      Search(TreeRoot, Numbers(I));
    END LOOP;
  END LOOP;
  NEW_LINE:
  PUT_LINE("DONE");
END MTPtr;
Listing 8: Source code for Ada Disk Write benchmark program.
with TEXT_IO;
use TEXT_IO;
Procedure MTWRITE is
Num_Rec : constant := 512;
Small: STRING(1..30);
Big : STRING(1..120);
F : FILE_TYPE;
BEGIN
 Small(1..30) := "123456781234567812345678123456";
 Big := Small & Small & Small & Small;
 CREATE(F, OUT_FILE, "A: TEMPO.DAT");
 FOR I in 1.. Num_Rec LOOP
   PUT_LINE(F, Big);
 END LOOP;
 CLOSE(F);
 PUT_LINE("DONE");
END MTWRITE;
Listing 9: Source code for Ada Disk Read benchmark program.
with TEXT_IO;
use TEXT_IO;
Procedure MTREAD is
Num_Rec : constant := 512;
Big : STRING(1..120);
Last: NATURAL;
F : FILE_TYPE;
```

```
BEGIN
```

```
OPEN(F, IN_FILE, "A:TEMPO.DAT");

FOR I in 1..Num_Rec LOOP
   GET_LINE(F, Big, Last);
END LOOP;
CLOSE(F);
PUT_LINE("DONE");
END MTREAD;
```

DHRY.C Dhrystone benchmark program by Reinhold P. Weicker, translated from Ada by Rick Richardson.
Listing accompanies the article "High-Tech Horsepower" by the BYTE editorial staff, July 1987, page 101. These are standard C language benchmarks used in BYTE reviews.

EVERBODY:

Please read "APOLOGY" below. -rick 01/06/85 See introduction in net.arch, or net.micro

"DHRYSTONE" Benchmark Program

Version:

C/1.1, 12/01/84

Date:

PROGRAM updated 01/06/86, RESULTS updated 03/31/86

Author:

Reinhold P. Weicker, CACM vol. 27, no. 10, 10/84, pg. 1013

Translated from Ada by Rick Richardson

Every method to preserve Ada likeness has been used, at the expense of C-ness.

Compile:

cc -0 dry.c -o drynr

: No registers

cc -O -DREG=register dry.c -o dryr

: Registers

Defines:

Defines are provided for old C compilers that don't have enums and can't assign structures. The time(2) function is library-dependent; most return the time in seconds, but beware of some, like Aztec C, which return other units. The LOOPS define is initially set for 50,000 loops. If you have a machine with large integers and is very fast, please change this number to 500,000 to get better accuracy. Please select the way to measure the execution time using the TIME define. For single-user machines, time(2) is adequate. For multiuser machines where you cannot get single-user access, use the times(2) function. If you have neither, use a stopwatch in the dead of night. Use a "printf" at the point marked "start timer" to begin your timings. DO NOT use the Unix time(1) command, as this will measure the total time to run this program, which will (erroneously) include the time to malloc(3) storage and to compute the time it takes to

do nothing.

Run:

drynr; dryr

Results:

If you get any new machine/OS results, please send to:

ihnp4!castor!pcrat!rick

and thanks to all that do. Space prevents listing the names of those who have provided some of these results. I'll be forwarding these results to Reinhold P. Weicker.

Note:

I order the list in increasing performance of the "with registers" benchmark. If the compiler doesn't provide register variables, then the benchmark is the same for both REG and NOREG.

PLEASE:

Send complete information about the machine type, clock speed, OS, and C manufacturer/version. If the machine is modified, tell me what was done. On Unix, execute uname -a and cc -V to get this info.

80x8x NOTE:

80x8x benchers: Please try to do all memory models for a particular compiler.

APOLOGY

(1/30/86):

Well, I goofed things up! As pointed out by Haakon Bugge, the line of code marked "GOOF" below was missing from the Dhrystone distribution for the last several months. It *WAS* in a backup copy I made last winter, so no doubt it was victimized by sleepy fingers operating vi!

The effect of the line missing is that the reported benchmarks are 15 percent too fast (at least on an 80286). Now, this creates a dilemma- -do I throw out ALL the data so far collected and use only results from this (corrected) version, or do I just keep collecting data for the old version?

Since the data collected so far *is* valid as long as it is compared with like data, I have decided to keep TWO lists--one for the old benchmark, and one for the new. This also gives me an opportunity to correct one other error I made in the instructions for this benchmark. My experience with C compilers has been mostly with Unix 'pcc'-derived compilers, where the 'optimizer' simply fixes sloppy code generation (peephole optimization). But today, there exist C compiler optimizers that will actually perform optimization in the computer science sense of the word, by removing, for example, assignments to a variable whose value is never used. Dhrystone, unfortunately, provides lots of opportunities for this sort of optimization.

I request that benchmarkers re-run this new, corrected version of Dhrystone, turning off or bypassing optimizers that perform more than peephole optimization. Please indicate the version of Dhrystone used when reporting the results to me.

* RESULTS BEGIN HERE

DHRYSTONE	VERSION 1	.1 RESULTS	BEGIN
-----------	-----------	------------	-------

MACHINE TYPE	MICROPROCESSOR	OPERATING SYSTEM	COMPILER	DHRYSTOI NO REG	NES/SEC. REGS
Apple IIe	65CO2-1.02 MHz	DOS 3.3	Aztec CII v1.05i	37	37
-	Z80-2.5 MHz	CPM-80 v2.2	Aztec CII v1.05g	91	91
-	8086-8 MHz	RMX86 V6	Intel C-86 V2.0	197	203LM??
IBM PC XT	8088-4.77 MHz	COHERENT 2.3.43	Mark Williams	259	275
-	8086-8 MHz	RMX86 V6	Intel C-86 V2.0	287	304 ??
Fortune 32:16	68000-6 MHz	V7+sys3+4.1BSD	cc	360	346
PDP-11/34A	w/FP-11C	UNIX V7m	cc	406	449
Macintosh 512	68000-7.7 MHz	Mac ROM O/S	DeSmet(C ware)	625	625
VAX-11/750	w/FPA	UNIX 4.2BSD	cc	831	852
DataMedia 932	68000-10 MHz	UNIX sysV	cc	837	888
Plexus P35	68000-12.5 MHz	UNIX sysIII	cc	835	894
ATT PC7300	68010-10 MHz	UNIX 5.0.3	cc	973	1034
Compaq II	80286-8 MHz	MSDOS 3.1	MS C 3.0	1086	1140 LM
IBM PC AT	80286-7.5 MHz	Venix/286 SVR2	cc	1159	1254 *15
Compaq II	80286-8 MHz	MSDOS 3.1	MS C 3.0	1190	1282 MM
MicroVAX II	_	Mach/4.3	cc	1361	1385
DEC uVAX II	may see 1 the o post summa leader	Ultrix-32m v1.1	cc Company of the control of	1385	1399
Compaq II	80286-8 MHz	MSDOS 3.1	MS C 3.0	1351	1428
VAX-11/780	ic to 1 128 Miller Liut fl soft	UNIX 4.2BSD	cc distance of the	1417	1441
VAX-780/MA780	rest, please chapus this p	Mach/4.3	CC CC COMM P SV AM LOC	1428	1470
VAX-11/780	Beren Fert verimosys and at	UNIX 5.0.1	cc 4.1.1.31	1650	1640
Ridge 32C V1	A MINTEL GOLDING JOSELLATION	ROS 3.3	Ridge C (older)	1628	1695
Gould PN6005	SCALE & THE LINCOLLINE SAND NO.	UTX 1.1c+ (4.2)	cc	1732	1884
Gould PN9080	custom ECL	UTX-32 1.1C	cc	4745	4992
VAX-784	TO ARE SUPERIOR BUT OF S	Mach/4.3	cc	5263	5555 &4
VAX 8600	a visual pur suppliment on pure	4.3 BSD	cc	6329	6423
Amdah1 5860	4	UTS sysV	cc 1.22	28,735	28,846
IBM3090/200	A KANA SHARMAN TO THE THE TANK	?	?	31,250	31,250

DHRYSTONE	VERSTON	1.0 RESUL	TS BEGIN .

MACHINE	MICROPROCESSOR	OPERATING	COMPILER	DHRYSTONI	
TYPE	raves one saud to a sessor are	SYSTEM	et shar track a contract	NO REG	REGS
Commodore 64	6510-1 MHz	C64 ROM	C Power 2.8	36	36
HP-110	8086-5.33 MHz	MS-DOS 2.11	Lattice 2.14	284	284
IBM PC XT	8088-4.77 MHz	PC/IX	cc	271	294
CCC 3205	March Transferred and Such State	Xelos(SVR2)	cc	558	592
Perq-II	2901 bitslice	Accent S5c	cc (CMU)	301	301
IBM PC XT	8088-4.77 MHz	COHERENT 2.3.43	Mark Williams cc	296	317
Cosmos	68000-8 MHz	UniSoft	cc	305	322
IBM PC XT	8088-4.77 MHz	Venix/862.0	cc	297	324
DEC PRO 350	11/23	Venix/PRO SVR2	cc	299	325
IBM PC	8088-4.77 MHz	MS-DOS 2.0	b16cc 2.0	310	340
PDP11/23	11/23	Venix (V7)	cc	320	358
Commodore Amiga		?	Lattice 3.02	368	371
PC XT	8088-4.77 MHz	Venix/86 SYS V	cc	339	377
IBM PC	8088-4.77 MHz	MS-DOS 2.0	CI-C86 2.20M	390	390
IBM PC XT	8088-4.77 MHz	PC-DOS 2.1	Wizard 2.1	367	403
IBM PC XT	8088-4.77 MHz	PC-DOS 3.1	Lattice 2.15	403	403 (
Colex DM-6	68010-8 MHz	Unisoft SYSV	cc	378	410
IBM PC	8088-4.77 MHz	PC-DOS 3.1	Datalight 1.10	416	416
IBM PC	NEC V20-4.77 MHz	MS-DOS 3.1	MS 3.1	387	420
IBM PC XT	8088-4.77 MHz	PC-DOS 2.1	Microsoft 3.0	390	427

4	* IBM PC	NEC V20-4.77 MHz	MS-DOS 3.1	MS 3.1 (186)	393	427
		NEC V20-4.77 PINZ			387	438
	PDP-11/34	-	Unix V7M	CC		
	F IBM PC	8088, 4.77 MHz	PC-DOS 2.1	Aztec C v3.2d	423	454
,	Tandy 1000	V20, 4.77 MHz	MS-DOS 2.11	Aztec C v3.2d	423	458
+	Tandy TRS-16B	68000-6 MHz	Xenix 1.3.5	cc	438	458
+	PDP-11/34		RSTS/E	decus c	438	495
+	Onyx C8002	Z8000-4 MHz	IS/11.1(V7)	cc	476	511
	Tandy TRS-16B	68000-6 MHz	Xenix 1.3.5	Green Hills	609	617
	DEC PRO 380	11/73	Venix/PRO SVR2	cc	577	628
		68000-10 MHz	0s9/68000	version 1.3	603	649 FH
	FHL QT+				666	666
	Apollo DN550	68010-? MHz	AegisSR9/IX	cc 3.12		
	HP-110	8086-5.33 MHz	MS-DOS 2.11	Aztec C	641	676
+	• ATT PC6300	8086-8 MHz	MS-DOS 2.11	b16cc 2.0	632	684
+	IBM PC AT	80286-6 MHz	PC-DOS 3.0	CI-C86 2.1	666	684
+	Tandy 6000	68000-8 MHz	Xenix 3.0	cc	694	694
	IBM PC AT	80286-6 MHz	Xenix 3.0	cc	684	704 MM
	Macintosh	68000-7.8 MHz 2M	Mac Rom	Mac C 32 bit int	694	704
			riac itom		661	709
	Macintosh	68000-7.7 MHz		MegaMax C 2.0		
	Macintosh 512	68000-7.7 MHz	Mac ROM O/S	DeSmet(C ware)	714	714
+	FIBM PC AT	80286-6 MHz	Xenix 3.0	cc	704	714 LM
+	Codata 3300	68000-8 MHz	UniPlus+ (v7)	cc	678	725
+	WICAT MB	68000-8 MHz	System V	WICAT C 4.1	585	731
	Cadmus 9000	68010-10 MHz	Unix	cc	714	735
	* AT&T 6300	8086-8 MHz	Venix/86 SVR2	cc	668	743
					720	747
	Cadmus 9790	68010-10 MHz 1MB	SVRO, Cadmus3.7	cc		
	NEC PC9801F	8086-8 MHz	PC-DOS 2.11	Lattice 2.15	768	- @
,	• ATT PC6300	8086-8 MHz	MS-DOS 2.11	CI-C86 2.20M	769	769
+	Burroughs XE550	68010-10 MHz	Centix 2.10	cc	769	769 CT1
+	EAGLE/TURBO	8086-8 MHz	Venix/86 SVR2	cc	696	779
	ALTOS 586	8086-10 MHz	Xenix 3.0b	cc	724	793
	DEC 11/73	J-11 micro	Ultrix-11 V3.0	cc	735	793
					735	806
	ATT 3B2/300	WE32000-? MHz	Unix 5.0.2	CC		
	Apollo DN320	68010-? MHz	AegisSR9/IX	cc 3.12	806	806
,	* IRIS-2400	68010-10 MHz	Unix System V	cc	772	829
+	Atari 520ST	68000-8 MHz	TOS	DigResearch	839	846
+	FIBM PC AT	80286-6 MHz	PC-DOS 3.0	MS 3.0(large)	833	847 LM
+	WICAT MB	68000-8 MHz	System V	WICAT C 4.1	675	853 S
	₹ VAX-11/750		Ultrix 1.1	4.2BSD cc	781	862
		68000-8 MHz	UniSoft V.2	cc	821	875
	CCC 7350A	08000-8 Filiz				877
	VAX-11/750	-	Unix 4.2bsd	cc	862	
	Fast Mac	68000-7.7 MHz		MegaMax C 2.0	839	904 +
)	* IBM PC XT	8086-9.54 MHz	PC-DOS 3.1	Microsoft 3.0	833	909 C1
+	DEC 11/44		Ultrix-11 V3.0	cc	862	909
+	Macintosh	68000-7.8 MHz 2M	Mac Rom	Mac C 16 bit int	877	909 S
	* CCC 3210		Xelos RO1(SVR2)	cc	849	924
	* CCC 3220	_	Ed. 7 v2.3	cc	892	925
	IBM PC AT	80286-6 MHz	Xenix 3.0	cc -i	909	925
					862	943
	* AT&T 6300	8086, 8 MHz	MS-DOS 2.11	Aztec C v3.2d		
	IBM PC AT	80286-6 MHz	Xenix 3.0	cc	892	961
)	* VAX-11/750	w/FPA	Eunice 3.2	cc	914	976
+	* IBM PC XT	8086-9.54 MHz	PC-DOS 3.1	Wizard 2.1	892	980 C1
+	IBM PC XT	8086-9.54 MHz	PC-DOS 3.1	Lattice 2.15	980	980 C1
	Plexus P35	68000-10 MHz	Unix System III	cc	984	980
	* PDP-11/73	KDJ11-AA 15 MHz	Unix V7M 2.1	cc	862	981
	VAX-11/750	w/FPA	Unix 4.3bsd	cc	994	997
					909	
	FIRIS-1400	68010-10 MHz	Unix System V	cc		1000
	FIBM PC AT	80286-6 MHz	Venix/862.1	cc	961	1000
	F IBM PC AT	80286-6 MHz	PC-DOS 3.0	b16cc 2.0	943	1063
)	* Zilog S8000/11	Z8001-5.5 MHz	Zeus 3.2	cc	1011	1084
+	NSC ICM-3216	NSC 32016-10 MHz	Unix SVR2	cc	1041	1084
	* IBM PC AT	80286-6 MHz	PC-DOS 3.0	MS 3.0(small)	1063	1086
	* VAX-11/750	w/FPA	VMS	VAX-11 C 2.0	958	1091
	Stride	68000-10 MHz	System-V/68	cc	1041	1111
	-11-	MC68000-12.5 MHz	Unix SYSIII	Plexus	1111	1111
	Flexus P/60					
	* ATT PC7300	68010-10 MHz	Unix 5.0.2	cc	1041	1111
	* CCC 3230	-	Xelos RO1(SVR2)	,cc	1040	1126
)	* Stride	68000-12 MHz	System-V/68	cc	1063	1136
+	* IBM PC AT	80286-6 MHz	Venix/286 SVR2	cc	1056	1149
	* Plexus P/60	MC68000-12.5 MHz	Unix SYSIII	Plexus	1111	1163 T
	* IBM PC AT	80286-6 MHz	PC-DOS 3.0	Datalight 1.10	1190	1190
	* ATT PC6300+	80286-6 MHz	MS-DOS 3.1	b16cc 2.0	1111	1219
	* IBM PC AT	80286-6 MHz	PC-DOS 3.1	Wizard 2.1	1136	1219
	* Sun 2/120	68010-10 MHz	Sun 4.2BSD	CC	1136	1219
	* IBM PC AT	80286-6 MHz	PC-DOS 3.0	CI-C86 2.20M	1219	1219
)	* WICAT PB	68000-8 MHz	System V	WICAT C 4.1	998	1226

July

* MASSCOMP 500	68010-10 MHz	RTU V3.0	cc (V3.2)	1156	1238
* Alliant FX/8	IP (68012-12 MHz)	Concentrix	cc -ip;exec -i	1170	1243 FX
* Cyb DataMate	68010-12.5 MHz	Uniplus 5.0	Unisoft cc	1162	1250
* PDP 11/70		Unix 5.2	cc	1162	1250
* IBM PC AT	80286-6 MHz	PC-DOS 3.1	Lattice 2.15	1250	1250
* IBM PC AT	80286-7.5 MHz	Venix/862.1	cc	1190	1315 *15
* Sun2/120	68010-10 MHz	Standalone	cc Millian	1219	1315
* Intel 380	80286-8 MHz	Xenix R3.Oup1	cc	1250	1315 *16
* Sequent Balance 8000	NS32032-10 MHz	Dynix 2.0	cc	1250	1315 N12
* IBM PC/DSI-32	32032-10 MHz	MS-DOS 3.1	Green Hills 2.14	1282	1315 C3
* ATT 3B2/400	WE32100-? MHz	Unix 5.2	cc	1315	1315
* CCC 3250XP		Xelos R01(SVR2)	cc	1215	1318
* IBM PC RT 032	RISC(801?)? MHz	BSD 4.2	cc	1248	1333 RT
* DG MV4000	-	AOS/VS 5.00	oc cc	1333	1333
* IBM PC AT	80286-8 MHz	Venix/862.1	nex cc	1275	1380 *16
* IBM PC AT	80286-6 MHz	MS-DOS 3.0	Microsoft 3.0	1250	1388
* ATT PC6300+	80286-6 MHz	MS-DOS 3.1	CI-C86 2.20M	1428	1428
* COMPAQ/286	80286-8 MHz	Venix/286 SVR2	cc	1326	1443
* IBM PC AT	80286-7.5 MHz	Venix/286 SVR2	cc and cc	1333	1449 *15
* WICAT PB	68000-8 MHz	System V	WICAT C 4.1	1169	1464 S
* Tandy II/6000	68000-8 MHz	Xenix 3.0	cc	1384	1477
* MicroVAX II		Mach/4.3	cc	1513	1536
* WICAT MB	68000-12.5 MHz	System V	WICAT C 4.1	1246	1537
* IBM PC AT	80286-9 MHz	SCO Xenix V	cc sime-asos	1540	1556 *18
* Cyb DataMate	68010-12.5 MHz	Uniplus 5.0	Unisoft cc	1470	1562 S
* VAX-11/780	_	Unix 5.2	cc 8HM 8-3808	1515	1562
* MicroVAX-II	2 / 10 / 10 / 10 / 10 / 10 / 10 / 10 / 1		38. s-	1562	1612
* VAX-780/MA780		Mach/4.3	cc	1587	1612
* VAX-11/780	_	Unix 4.3bsd	cc	1646	1662
* Apollo DN660	_	AegisSR9/IX	cc 3.12	1666	1666
* ATT 3B20	_	Unix 5.2	cc	1515	1724
* NEC PC-98XA	80286-8 MHz	PC-DOS 3.1	Lattice 2.15	1724	1724@
* HP9000-500	B series CPU	HP-UX 4.02	cc	1724	-
* Ridge 32C V1	- 3	ROS 3.3	Ridge C (older)	1776	-0040-01
* IBM PC/STD	80286-8 MHz	MS-DOS 3.0	Microsoft 3.0	1724	1785 C2
* WICAT MB	68000-12.5 MHz	System V	WICAT C 4.1	1450	1814 S
* WICAT PB	68000-12.5 MHz	System V	WICAT C 4.1	1530	1898
* DEC-2065	KL10-Model B	TOPS-20 6.1FT5	Port. C Comp.	1937	1946
* Gould PN6005	- House I	UTX 1.1(4.1BSD)	cc semp.	1675	1964
* DEC2060	KL-10	TOPS-20	lad cc	2000	2000 NM
* Intel 310AP	80286-8 MHz	Xenix 3.0	CC	1893	2009
* VAX-11/785	-	Unix 5.2	-09 cc 11 1 MM 48 9-88681	2083	2083
* VAX-11/785		VMS	VAX-11 C 2.0	2083	2083
* VAX-11/785	a made a tot surface	Unix SVR2	CC MS	2123	2083
* VAX-11/785		ULTRIX-32 1.1	cc	2083	2091
* VAX-11/785		Unix 4.3bsd	cc	2135	2136
	68000-12.5 MHz		WICAT C 4.1	1780	
* WICAT PB * Pyramid 90x	08000-12.) MHZ	System V	CC CC	2272	2233 S 2272
* Pyramid 90x	FPA, cache, 4Mb	0Sx 2.3 0Sx 2.5	cc no -0	2777	2777
0	w/cache		cc w/-0		
* Pyramid 90x * IBM-4341-II	w/cache	0Sx 2.5		3333	3333
	68000 16 60 MU-	VM/SP3	Waterloo C 1.2	3333	3333
* IRIS-2400T	68020-16.67 MHz	Unix System V	cc	3105	3401
* Celerity C-1200	?	Unix 4.2BSD	cc	3485	3468
* SUN 3/75	68020-16.67 MHz	SUN 4.2 V3	cc .	3333	3571
* IBM-4341	Model 12	UTS 5.0	?	3685	3685
* SUN 3/160	68020-16.67 MHz	Sun 4.2 V3.0A	cc	3381	3764
* Sun 3/180	68020-16.67 MHz	Sun 4.2	cc	3333	3846
* IBM-4341	Model 12	UTS 5.0	?	3910	3910 MN
* MC 5400	68020-16.67 MHz	RTU V3.0	cc (V4.0)	3952	4054
* Intel 386/20	80386-12.5 MHz	PMON debugger	Intel C386v0.2	4149	4386
* NCR Tower32	68020-16.67 MHz	SYS 5.0 Rel 2.0	cc	3846	4545
* MC 5600/5700	68020-16.67 MHz	RTU V3.0	cc (V4.0)	4504	4746 %
* Intel 386/20	80386-12.5 MHz	PMON debugger	Intel C386v0.2	4534	4794 11
* Intel 386/20	80386-16 MHz	PMON debugger	Intel C386v0.2	5304	5607
* Gould PN9080	custom ECL	UTX-32 1.1C	cc	5369	5676
* Gould 1460-342	ECL proc	UTX/32 1.1/c	cc	5342	5677 G1
* VAX-784	-	Mach/4.3	cc	5882	5882 &4
* Intel 386/20	80386-16 MHz	PMON debugger	Intel C386v0.2	5801	6133 11
* VAX 8600	-	Unix 4.3bsd	cc	7024	7088
* VAX 8600	-	VMS	VAX-11 C 2.0	7142	7142
* Alliant FX/8	CE '	Concentrix	cc -ce;exec -c	6952	7655 FX
* CCI POWER 6/32		COS(SV+4.2)	cc	7500	7800
* CCI POWER 6/32		POWER 6 Unix/V	cc	8236	8498
* CCI POWER 6/32		4.2 Rel. 1.2b	cc	8963	9544
* Sperry (CCI Power 6)		4.2BSD	cc	9345	10,000
* CRAY-X-MP/12	105 MHz	COS 1.14	Cray C	10,204	10,204
and the same of th					

```
* IBM-3083
                                                        UTS 5.0 Rel 1
                                                                                                             16,666
                                                                                  cc
                                                                                                                        12,500
   CRAY-1A
                             80 MHz
                                                        CTSS
                                                                                  Cray C 2.0
                                                                                                             12,100
                                                                                                                        13,888
   IBM-3083
                                                        VM/CMS HPO 3.4
                                                                                  Waterloo C 1.2
                                                                                                             13,889
                                                                                                                        13,889
   Amdahl 470 V/8
                                                        UTS/V 5.2
                                                                                  cc v1.23
                                                                                                             15,560
                                                                                                                        15,560
   CRAY-X-MP/48
                             105 MHz
                                                        CTSS
                                                                                  Cray C 2.0
                                                                                                             15,625
                                                                                                                        17,857
   Amdahl 580
                                                        UTS 5.0 Rel 1.2
                                                                                  cc v1.5
                                                                                                             23,076
                                                                                                                        23,076
   Amdah1 5860
                                                       UTS/V 5.2
                                                                                  cc v1.23
                                                                                                             28,970
                                                                                                                        28,970
 ¥
   NOTE
          Crystal changed from 'stock' to listed value.
          This Macintosh was upgraded from 128K to 512K in such a way that the new 384K of memory is not slowed down by video
    +
          generator accesses.
          Single processor: MC == MASSCOMP.
    MM
          A version 7 C compiler written at New Mexico Tech.
          Vanilla Lattice compiler used with MicroPro standard library.
    S
          Shorts used instead of ints.
    T
          With Chris Torek's patches (whatever they are).
          For WICAT Systems: MB=MultiBus, PB=Proprietary Bus.
          Large Memory Model. (Otherwise, all 80x8x results are small model).
    MM
          Medium Memory Model. (Otherwise, all 80x8x results are small model).
    C1
          Univation PC TURBO Coprocessor; 9.54-MHz 8086, 640K RAM
    02
          Seattle Telecom STD-286 board.
    C3
          Definicon DSI-32 coprocessor.
    C?
          Unknown coprocessor board.
    CT1
          Convergent Technologies MegaFrame, 1 processor.
          Using Mike Newton's 'optimizer' (see net.sources).
    MN
         This Gould machine has two processors and was able to run two Dhrystone benchmarks in parallel with
          no slowdown.
          FHC == Frank Hogg Labs (Hazelwood Uniquad 2 in an FHL box).
    FX
         The Alliant FX/8 is a system consisting of 1-8 CEs (computation engines) and 1-12 IPs (interactive processors). Note
         N8 applies.
    RT
         This is one of the RTs that CMU has been using for awhile. I'm not sure that this is identical to the machine that IBM is
          selling to the public.
         Normally, the 386/20 starter kit has a 16K direct-mapped cache, which inserts two or three wait states on a write-
          through. These results were obtained by disabling the write-through, or essentially turning the cache into
          zero-wait-state memory.
         This machine has multiple processors, allowing "nn" copies of the benchmark to run in the same time as one copy.
         This machine has "nn" processors, and the benchmark results were obtained by having all "nn" processors working on one
         copy of Dhrystone. (Note this is different than Nnn. Salesmen like this measure.)
         I don't trust results marked with '?'. These were sent to me with either incomplete info, or with times that just don't
         make sense. ?? means I think the performance is too poor; ?! means too good. If anybody can confirm these figures,
         please respond.
   ABBREVIATIONS
         CCC
                   Concurrent Computer Corp. (was Perkin-Elmer)
         MC
                   Masscomp
                                                  ----- RESULTS END -----
         The following program contains statements of a high-level programming language (C) in a distribution
         considered representative:
         assignments
                                            53%
                                           32%
         control statements
         procedure, function calls
                                           15%
         100 statements are dynamically executed. The program is balanced with respect to the three aspects:
                   - statement type
                   - operand type (for simple data types)
                   - operand access
                             operand global, local, parameter, or constant.
         The combination of these three aspects is balanced only approximately.
         The program does not compute anything meaningful, but it is syntactically and semantically correct.
/* Accuracy of timings and human fatigue controlled by next two lines */
#define LOOPS
                  50000
                                        /* Use this for slow or 16-bit machines */
*#define LOOPS
                  500000
                                         /* Use this for faster machines */
```

```
/* Compiler-dependent options */
                                       /* Define if compiler has no enums */
#undef NOENUM
#undef NOSTRUCTASSIGN
                                       /* Define if compiler can't assign structures */
/* define only one of the next two defines */
                                       /* Use times(2) time function */
/*#define TIMES
#define TIME
                                       /* Use time(2) time function */
/* define the granularity of your times(2) function (when used) */
                                        /* times(2) returns 1/60 second (most) */
#define HZ
               60
                                        /* times(2) returns 1/100 second (WECo) */
/*#define HZ
               100
/* for compatibility with goofed-up version */
                                       /* Define if you want the goofed-up version */
/*#define GOOF
#ifdef GOOF
char
         Version[] = "1.0";
#else
char
         Version[] = "1.1";
#endif
#ifdef
         NOSTRUCTASSIGN
                                       memcpy(&(d), &(s), sizeof(d))
#define structassign(d, s)
#else
#define
        structassign(d, s)
                                        d = s
#endif
         NOENUM
#ifdef
#define
         Ident1
#define Ident2
                   2
#define Ident3
#define Ident4
#define Ident5
typedef int
                   Enumeration;
#else
                   {Ident1, Ident2, Ident3, Ident4, Ident5} Enumeration;
typedef enum
#endif
typedef int
                   OneToThirty;
typedef int
                   OneToFifty;
typedef char
                   CapitalLetter;
typedef char
                   String30[31];
typedef int
                   Array1Dim[51];
typedef int
                   Array2Dim[51][51];
            Record
struct
             struct Record
                                           *PtrComp;
                                           Discr;
             Enumeration
             Enumeration
                                           EnumComp;
             OneToFifty
                                           IntComp;
             String30
                                           StringComp;
};
typedef struct Record RecordType;
typedef RecordType * RecordPtr;
typedef int
                       boolean:
              NULL
                            0 */
/* #define
#define TRUE
                      1
#define FALSE
                      0
#ifndef REG
#define
         REG
#endif
                                Func1();
extern Enumeration
extern boolean
                                Func2();
#include <HD20:C:#include files:stdio.h>
#ifdef TIMES
#include <HD20:C:#include files:unix #includes:types.h>
#include <HD20:C:#include files:unix #includes:time.h>
#endif
```

```
main()
       Proc0():
       exit(0);
  * Package 1
  */
 int
                              IntGlob;
 boolean
                              BoolGlob;
 char
                              Char1Glob;
 char
                              Char2Glob;
 Array1Dim
                              Array1Glob;
 Array2Dim
                              Array2Glob;
 RecordPtr
                              PtrGlb;
RecordPtr
                             PtrGlbNext:
 Proc0()
                 OneToFifty
                                                    IntLoc1;
                 REG OneToFifty
                                                    IntLoc2;
                OneToFifty
                                                    IntLoc3;
                 REG char
                                                    CharLoc;
                REG char
                                                    CharIndex;
                Enumeration
                                                    EnumLoc;
                String30
                                                    String1Loc;
                String30
                                                    String2Loc;
                extern char
                                                    *malloc();
#ifdef TIME
                long
                                                    time();
                long
                                                    starttime;
                long
                                                    benchtime;
                long
                                                    nulltime:
                register unsigned int
                                                    i;
                starttime = time( (long *) 0);
                for (i = 0; i < LOOPS; ++i);
                nulltime = time( (long *) 0) - starttime; /* Computes o'head of loop */
#endif
#ifdef TIMES
                time_t
                                                   starttime;
                time_t
                                                   benchtime;
                time_t
                                                   nulltime;
                struct tm
                                                    tms;
                register unsigned int
                                                    i:
                times(&tms); starttime = tms.tms_utime;
                for (i = 0; i < LOOPS; ++i);
                times(&tms);
                nulltime = tms.tms_utime - starttime; /* Computes overhead of looping */
#endif
                PtrGlbNext = (RecordPtr) malloc(sizeof(RecordType));
                PtrGlb = (RecordPtr) malloc(sizeof(RecordType));
                PtrGlb->PtrComp = PtrGlbNext;
                PtrGlb->Discr = Ident1;
                PtrGlb->EnumComp = Ident3;
                PtrGlb->IntComp = 40;
                strepy(PtrGlb->StringComp, "DHRYSTONE PROGRAM, SOME STRING");
#ifndef GOOF
                strepy(String1Loc, "DHRYSTONE PROGRAM, 1ST STRING"); /*GOOF*/
#endif
                {\tt Array2Glob[8][7] = 10; /* Was missing in published program */ }
/***************
 -- Start Timer --
****************
#ifdef TIME
         starttime = time( (long *) 0);
#endif
#ifdef TIMES
         times(&tms); starttime = tms.tms_utime;
```

```
#endif
         for (i = 0; i < LOOPS; ++i)
                   Proc5();
                   Proc4();
                   IntLoc1 = 2;
                   IntLoc2 = 3;
                   strepy(String2Loc, "DHRYSTONE PROGRAM, 2ND STRING");
                   EnumLoc = Ident2;
                   BoolGlob = ! Func2(String1Loc, String2Loc);
                   while (IntLoc1 < IntLoc2)
                             IntLoc3 = 5 * IntLoc1 - IntLoc2;
                             Proc7(IntLoc1, IntLoc2, &IntLoc3);
                             ++IntLoc1;
                   Proc8(Array1Glob, Array2Glob, IntLoc1, IntLoc3);
                   Proc1(PtrGlb);
                   for (CharIndex = 'A'; CharIndex <= Char2Glob; ++CharIndex)</pre>
                            if (EnumLoc == Func1(CharIndex, 'C'))
                                     Proc6(Ident1, &EnumLoc);
                   IntLoc3 = IntLoc2 * IntLoc1;
                   IntLoc2 = IntLoc3 / IntLoc1;
                   IntLoc2 = 7 * (IntLoc3 - IntLoc2) - IntLoc1;
                   Proc2(&IntLoc1);
 -- Stop Timer --
#ifdef TIME
          benchtime = time( (long *) 0) - starttime - nulltime;
         printf("Dhrystone(%s) time for %ld passes = %l \ n",
                   Version,
                   (long) LOOPS, benchtime);
          printf("This machine benchmarks at %ld Dhrystones/second \ n",
                   ((long) LOOPS) / benchtime);
#endif
#ifdef TIMES
          times(&tms);
          benchtime = tms.tms_utime - starttime - nulltime;
          printf("Dhrystone(%s) time for %ld passes = %ld \ n",
                   Version,
                    (long) LOOPS, benchtime/HZ);
          printf("This machine benchmarks at %ld Dhrystones/second \ n",
                   ((long) LOOPS) * HZ / benchtime);
#endif
Proc1(PtrParIn)
REG RecordPtr
                PtrParIn;
                            (*(PtrParIn->PtrComp))
 #defineNextRecord
          structassign(NextRecord, *PtrGlb);
          PtrParIn->IntComp = 5;
          NextRecord.IntComp = PtrParIn->IntComp;
          NextRecord.PtrComp = PtrParIn->PtrComp;
          Proc3(NextRecord.PtrComp);
          if (NextRecord.Discr == Ident1)
                    NextRecord.IntComp = 6;
                    Proc6(PtrParIn->EnumComp, &NextRecord.EnumComp);
                    NextRecord.PtrComp = PtrGlb->PtrComp;
                    Proc7(NextRecord.IntComp, 10, &NextRecord.IntComp);
          else
                    structassign(*PtrParIn, NextRecord);
 #undef
          NextRecord
```

```
Proc2(IntParIO)
OneToFifty
                   *IntParIO;
          REG OneToFifty
                                    IntLoc;
          REG Enumeration
                                    EnumLoc;
          IntLoc = *IntParIO + 10;
          for(;;)
                          if (Char1Glob == 'A')
                                 --IntLoc;
                                 *IntParIO = IntLoc - IntGlob;
                                EnumLoc = Ident1;
                          if (EnumLoc == Ident1)
                                break;
          }
Proc3(PtrParOut)
RecordPtr
               *PtrParOut;
          if (PtrGlb != NULL)
                *PtrParOut = PtrGlb->PtrComp;
          else
                IntGlob = 100;
          Proc7(10, IntGlob, &PtrGlb->IntComp);
Proc4()
{
          REG boolean
                         BoolLoc;
          BoolLoc = Char1Glob == 'A';
          BoolLoc = BoolGlob;
          Char2Glob = 'B';
}
Proc5()
          Char1Glob = 'A';
          BoolGlob = FALSE;
extern boolean Func3();
Proc6(EnumParIn, EnumParOut)
REG Enumeration EnumParIn;
REG Enumeration *EnumParOut;
          *EnumParOut = EnumParIn;
          if (! Func3(EnumParIn))
                *EnumParOut = Ident4;
          switch (EnumParIn)
          case Ident1:
                          *EnumParOut = Ident1; break;
          case Ident2:
                          if (IntGlob > 100) *EnumParOut = Ident1;
                          else *EnumParOut = Ident4;
                          break;
          case Ident3:
                          *EnumParOut = Ident2; break;
         case Ident4:
                          break;
                          *EnumParOut = Ident3;
         case Ident5:
Proc7(IntParI1, IntParI2, IntParOut)
             IntParI1;
OneToFifty
OneToFifty
                IntParI2;
OneToFifty
                *IntParOut;
         REG OneToFifty IntLoc;
         IntLoc = IntParI1 + 2;
         *IntParOut = IntParI2 + IntLoc;
```

```
July
```

```
Proc8(Array1Par, Array2Par, IntParI1, IntParI2)
Array1Dim
                Array1Par;
Array2Dim
                 Array2Par;
OneToFifty
                 IntParI1;
                 IntParI2;
OneToFifty
         REG OneToFifty IntLoc;
         REG OneToFifty IntIndex;
         IntLoc = IntParI1 + 5;
          Array1Par[IntLoc] = IntParI2;
         Array1Par[IntLoc+1] = Array1Par[IntLoc];
          Array1Par[IntLoc+30] = IntLoc;
          for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)</pre>
                Array2Par[IntLoc][IntIndex] = IntLoc;
         ++Array2Par[IntLoc][IntLoc-1];
          Array2Par[IntLoc+20][IntLoc] = Array1Par[IntLoc];
          IntGlob = 5;
Enumeration Func1(CharPar1, CharPar2)
CapitalLetter CharPar1;
CapitalLetter CharPar2;
                                   CharLoc1;
         REG CapitalLetter
         REG CapitalLetter
                                   CharLoc2;
         CharLoc1 = CharPar1;
         CharLoc2 = CharLoc1;
          if (CharLoc2 != CharPar2)
                return (Ident1);
         else
                return (Ident2);
boolean Func2(StrParI1, StrParI2)
String30
                StrParI1;
                StrParI2;
String30
{
          REG OneToThirty
                                   IntLoc;
          REG CapitalLetter
                                    CharLoc;
          IntLoc = 1;
          while (IntLoc <= 1)
                if (Func1(StrParI1[IntLoc], StrParI2[IntLoc+1]) == Ident1)
                       CharLoc = 'A';
                      ++IntLoc;
          if (CharLoc >= 'W' && CharLoc <= 'Z')
                IntLoc = 7;
          if (CharLoc == 'X')
                return(TRUE);
          else
                if (stremp(StrParI1, StrParI2) > 0)
                       IntLoc += 7;
                       return (TRUE);
                else
                      return (FALSE);
boolean Func3(EnumParIn)
REG Enumeration EnumParIn;
          REG Enumeration EnumLoc;
          EnumLoc = EnumParIn;
          if (EnumLoc == Ident3) return (TRUE);
          return (FALSE);
```

```
#ifdef NOSTRUCTASSIGN
memcpy(d, s, 1)
register char    *d;
register char    *s;
register int    1;
{
          while (1--) *d++ = *s++;
}
#endif
```

FIB.C Accompanies the article "High-Tech Horsepower" by the BYTE editorial staff, July 1987, page 101. These are standard C language benchmarks used in BYTE reviews.

```
#include <stdio.h>
#define NTIMES 10/* number of times to compute Fibonacci value */
#define NUMBER 24/* biggest one we can compute with 16 bits */
main()
         /* compute Fibonacci value */
    int i;
    unsigned value, fib();
    printf("%d iterations: ", NTIMES);
    for (i = 1; i <= NTIMES; i++)
           value = fib(NUMBER);
    printf("Fibonacci(%d) = %u.\n", NUMBER, value);
    exit(0);
unsigned fib(x)
                   /* compute Fibonacci number recursively */
int x;
     if (x > 2)
           return (fib(x-1) + fib(x-2));
    else
           return (1);
```

FLOAT.C Accompanies the article "High-Tech Horsepower" by the BYTE editorial staff, July 1987, page 101. These are standard C language benchmarks used in BYTE reviews.

```
c = c/a;
c = a * b;
c = c/a;
printf ("Done\n");
```

SORT.C Accompanies the article "High-Tech Horsepower" by the BYTE editorial staff, July 1987, page 101. These are standard C language benchmarks used in BYTE reviews.

```
/* sorting benchmark--calls randomly the number of times specified by
  MAXNUM to create an array of long integers, then does a quicksort
  on the array of longs. The program does this for the number of
  times specified by COUNT.
#include "stdio.h"
#define MAXNUM 1000
#define COUNT 10
#define MODULUS ((long) 0x20000)
#define C 13849L
#define A 25173L
long seed = 7L;
long random();
long buffer [MAXNUM] = {0};
main()
       inti, j;
       long temp;
#include "startup.c"
       printf ("Filling array and sorting %d times\n", COUNT);
       for (i = 0; i < COUNT; ++i)
             for (j = 0; j < MAXNUM; ++j)
                    temp = random (MODULUS);
                          if (temp < OL)
                                 temp = (-temp);
                          buffer[j] = temp;
             printf ("Buffer full, iteration %d\n", i);
             quick (0, MAXNUM - 1, buffer);
#include "done.c"
quick (lo, hi, base)
       int lo, hi;
       long base [];
```

```
inti, j;
      long pivot, temp;
      if (lo < hi)
             for (i = lo, j = hi-1, pivot = base [hi]; i < j; )
                    while (i < hi && base [i] <= pivot)
                          ++1;
                    while (j > lo && base [j] >= pivot)
                           --j;
                    if (i < j)
                           temp = base [i];
                           base [i] = base [j];
                           base [j] = temp;
             temp = base [i];
             base [i] = base [hi];
             base [hi] = temp;
             quick (lo, i-1, base);
             quick (i + 1, hi, base);
long random (size)
    long size;
    seed = seed * A + C;
    return (seed % size);
```

SIEVE.C Accompanies the article "High-Tech Horsepower" by the BYTE editorial staff, July 1987, page 101. These are standard C language benchmarks used in BYTE reviews.

```
Eratosthenes Sieve prime-number program in from BYTE January 1983
#define TRUE 1
#define FALSE 0
#define size 8190
 char flags [size + 1];
main()
  int i, prime, k, count, iter;
  printf ("10 iterations\n");
  for (iter = 1; iter <= 10; iter++)
                                             /* do program 10 times */
       count = 0;
                                           /* prime counter */
       for (i = 0; i <= size; i++)
                                             /* set all flags true */
          flags [i] = TRUE;
       for (i = 0; i <= size; i++)
             if (flags [i])
                                           /* found a prime */
             prime = i + i + 3;
                                             /* twice index + 3 */
                                             /* printf ("\n%d", prime); */
              for (k = i + prime; k <= size; k+= prime)
                                                  /* kill all multiple */
                       flags [k] = FALSE;
             count++;
                                                    /* primes found */
 printf ("%d primes.\n", count);
                                          /* primes found on 10th pass */
```

SAVAGE.C Accompanies the article "High-Tech Horsepower" by the BYTE editorial staff, July 1987, page 101. These are standard C language benchmarks used in BYTE reviews.

```
** savage.c -- floating-point speed and accuracy test. C version
** derived from BASIC version that appeared in Dr. Dobb's Journal,
** Sept. 1983, pp. 120-122.
#define
           ILOOP
           2500
extern
            double
            tan(), atan(), exp(), log(), sqrt();
main()
int i;
double a;
            printf("start\n");
            a = 1.0;
            for (i = 1; i <= (ILOOP - 1); i++)
            a = tan(atan(exp(log(sqrt(a*a))))) + 1.0;
            printf("a = %20.14e\n", a);
           printf("done\n");
}
```

FILEIO.C Accompanies the article "High-Tech Horsepower" by the BYTE editorial staff, July 1987, page 101. These are standard C language benchmarks used in BYTE reviews.

```
/* file reading and writing benchmark
  sequentially writes a 65,000-byte file on disk
  generates random long numbers
  uses these modulo 65,000 to read and write strings of ODDNUM bytes
  with the file-handling system of the c package
  the random-number generator is set to a specific seed,
  so that all compilers should generate the same code
#define ERROR -1
#define READERR O
#define BEG 0
#define CURR 1
#define END 2
#define READ O
#define WRITE 1
#define UPDATE 2
#define OKCLOSE O
#define FILESIZE 65000L
#define COUNT 500
#define C 13849L
#define A 25173L
#define ODDNUM 23
long seed = 7L;
long random (), lseek ();
main ()
    int i;
    long j, pos;
    int fd;
    char buffer [ODDNUM + 1];
```

```
if ((fd = creat ("test.dat", WRITE)) == ERROR)
            abort ("Can't create data file\n");
     else printf("File opened for sequential writing\n");
     for (j = 0; j < FILESIZE; ++j)
            if (write(fd, "x", 1) == ERROR)
                  abort ("Unexpected EOF in writing data file\n");
     if (close (fd) != OKCLOSE)
            abort ("Error closing data file\n");
     else
            printf ("Normal termination writing data file\n");
     if ((fd = open ("test.dat", UPDATE)) == ERROR)
            abort ("Can't open data file for random reading and writing\n");
     else printf ("File opened for random reading and writing\n");
     for (i = 0; i < COUNT; ++i)
            j = random (FILESIZE);
            if (j < OL)
                  j = (-j);
            if (FILESIZE - j < ODDNUM)
                  continue:
            if ((pos = lseek (fd, j, BEG)) == -1L)
                  abort ("Error reading at random offset\n");
            if (read (fd, buffer, ODDNUM) == READERR)
                  abort ("Error reading at random offset\n");
           j = random (FILESIZE);
           if (j < OL)
                  j = (-j);
           if (FILESIZE - j < ODDNUM)
                  continue;
           if ((pos = lseek (fd, j, BEG)) == -1L)
                  abort ("Error seeking to random offset\n");
           if (write (fd, buffer, ODDNUM) == READERR)
                  abort ("Error writing at random offset\n");
     if (close (fd) != OKCLOSE)
           abort ("Error closing data file\n");
     else
           printf ("Normal termination from random reading and writing\n");
long random (size)
    long size;
    seed = seed * A + C;
    return (seed % size);
abort (message)
    char *message;
    printf (message);
    exit (ERROR);
```

WHET.C Accompanies the article "High-Tech Horsepower" by the BYTE editorial staff, July 1987, page 101. These are standard C language benchmarks used in BYTE reviews.

```
/*
From hplabs!sdcrdcf!sdcsvax!dcdwest!ittatc!decvax!mcnc!rti-sel!scirtp!dfh
Sun Aug 25 12:55:29 1985
Relay-Version: version B 2.10.2 9/18/84; site amdahl.UUCP
Posting-Version: version B 2.10.2 9/5/84; site scirtp.UUCP
Path: amdahl!hplabs!sdcrdcf!sdcsvax!dcdwest!ittatc!decvax!mcnc!rti-sel!scirtp!dfh
From: dfh@scirtp.UUCP (David F. Hinnant)
Newsgroups: net.sources
Subject: Whetstone benchmark source in C - enclosed
Message-ID: <353@scirtp.UUCP>
Date: 25 Aug 85 19:55:29 GMT
Date Received: 27 Aug 85 08:15:18 GMT
Distribution: net
Organization: SCI Systems, Research Triangle Park, NC
Lines: 252
```

```
Enclosed below is a C translation of the famous "Whetstone benchmark"
from the original Algol version. I have inserted printf() as a
compiler option. I think this translation is accurate. The only
listing accompanies the article "High-Tech Horsepower" by the BYTE editorial staff, July 1987, page 101. These are standard C
language benchmarks used in BYTE reviews.
Numbers I have to compare with are from an old Ridge-32 machine, and
these are from a Pascal translation (I caught one error in their
translation). If anyone has any numbers from FORTRAN, Pascal, or Algol
versions of the Whetstone, I would very much like to see them.
  David Hinnant
  SCI Systems Inc.
  {decvax, akgua}!mcnc!rti-sel!scirtp!dfh
P.S. There is a .signature file at the end of the listing. */
//======
1*
    Whetstone benchmark in C. This program is a translation of the
    original Algol version in "A Synthetic Benchmark" by H. J. Curnow
    and B. A. Wichman in Computer Journal, vol. 19, no. 1, February 1976.
 * Used to test compiler optimization and floating-point performance.
 * Compile by: cc -O -s -o whet whet.c
 * or: cc -O -DPOUT -s -o whet whet.c
    if output is desired.
#define ITERATIONS 10 /* 1 million Whetstone instructions */
#include "sane.h"
#include "stdio.h"
// #Options G H
#define POUT
#define Pout(n, j, k, x1, x2, x3, x4) timingOff;\
pout(n, j, k, x1, x2, x3, x4); timingOn
#define double extended
#define tickCount *((long *)0x16A)
#define timingOn ticks -= tickCount
#define timingOff ticks += tickCount
double xx1, xx2, xx3, xx4, x, y, z, t, t1, t2;
double e1[4];
int i, j, k, 1, n1, n2, n3, n4, n6, n7, n8, n9, n10, n11;
long ticks;
main()
 printf("\nWhetstone Benchmark\n\n");
 ticks = 0;
  timingOn;
  /* initialize constants */
  t = 0.499975:
  t1 = 0.50025;
  t2 = 2.0;
  /* set values of module weights */
 n1 = 0 * ITERATIONS;
 n2 = 12 * ITERATIONS;
 n3 = 14 * ITERATIONS;
 n4 = 345 * ITERATIONS;
 n6 = 210 * ITERATIONS;
 n7 = 32 * ITERATIONS;
 n8 = 899 * ITERATIONS;
 n9 = 616 * ITERATIONS;
 n10 = 0 * ITERATIONS;
 n11 = 93 * ITERATIONS;
/* MODULE 1: simple identifiers */
```

```
xx1 = 1.0;
   xx2 = xx3 = xx4 = -1.0;
   for(i = 1; i <= n1; i += 1) {
   xx1 = (xx1 + xx2 + xx3 - xx4) * t;
    xx2 = (xx1 + xx2 - xx3 - xx4) * t;
    xx3 = (xx1 - xx2 + xx3 + xx4) * t;
    xx4 = (-xx1 + xx2 + xx3 + xx4) * t;
 #ifdef POUT
  Pout(n1, n1, n1, xx1, xx2, xx3, xx4);
 /* MODULE 2: array elements */
  e1[0] = 1.0;
  e1[1] = e1[2] = e1[3] = -1.0;
  for (i = 1; i <= n2; i +=1) {
   e1[0] = (e1[0] + e1[1] + e1[2] - e1[3]) * t;
   e1[1] = (e1[0] + e1[1] - e1[2] + e1[3]) * t;
   e1[2] = (e1[0] - e1[1] + e1[2] + e1[3]) * t;
   e1[3] = (-e1[0] + e1[1] + e1[2] + e1[3]) * t;
 #ifdef POUT
 Pout(n2, n3, n2, e1[0], e1[1], e1[2], e1[3]);
 #endif
 /* MODULE 3: array as parameter */
  for (i = 1; i <= n3; i += 1)
   pa(e1);
#ifdef POUT
 Pout(n3, n2, n2, e1[0], e1[1], e1[2], e1[3]);
/* MODULE 4: conditional jumps */
  j = 1;
  for (i = 1; i <= n4; i += 1) {
    if (j == 1)
     j = 2;
    else
     j = 3;
    if (j > 2)
    j = 0;
    else
     j = 1;
   if (j < 1)
    j = 1;
    else
     j = 0;
#ifdef POUT
 Pout(n4, j, j, xx1, xx2, xx3, xx4);
#endif
/* MODULE 5: omitted */
/* MODULE 6: integer arithmetic */
 j = 1;
 k = 2;
 1 = 3;
  for (i = 1; i \le n6; i += 1) {
  j = j * (k - j) * (1 - k);

k = 1 * k - (1 - j) * k;
   1 = (1 - k) * (k + j);
   e1[1-2] = j + k + 1; /* C arrays are zero-based */
   e1[k-2] = j * k * 1;
```

```
#ifdef POUT
 Pout(n6, j, k, e1[0], e1[1], e1[2], e1[3]);
#endif
/* MODULE 7: trig. functions */
 x = y = 0.5;
 for(i = 1; i <= n7; i +=1) {
  x = t * atan(t2*sin(x)*cos(x)/(cos(x+y)+cos(x-y)-1.0));
  y = t * atan(t2*sin(y)*cos(y)/(cos(x+y)+cos(x-y)-1.0));
#ifdef POUT
 Pout(n7, j, k, x, x, y, y);
#endif
/* MODULE 8: procedure calls */
 x = y = z = 1.0;
 for (i = 1; i <= n8; i +=1)
  p3(x, y, &z);
#ifdef POUT
 Pout(n8, j, k, x, y, z, z);
#endif
/* MODULE 9: array references */
 j = 1;
 k = 2;
 1 = 3;
 e1[0] = 1.0;
 e1[1] = 2.0;
 e1[2] = 3.0;
 for(i = 1; i <= n9; i += 1)
  p0();
#ifdef POUT
 Pout(n9, j, k, e1[0], e1[1], e1[2], e1[3]);
#endif
/* MODULE 10: integer arithmetic */
 j = 2;
 k = 3;
 for(i = 1; i <= n10; i +=1) {
  j = j + k;
  k = j + k;
  j = k - j;
  k = k - j - j;
#ifdef POUT
 Pout(n10, j, k, xx1, xx2, xx3, xx4);
#endif
/* MODULE 11: standard functions */
 x = 0.75;
 for(i = 1; i <= n11; i +=1)
  x = sqrt(exp(log(x) / t1));
#ifdef POUT
 Pout(n11, j, k, x, x, x, x);
#endif
 timingOFF;
 printf("\nWhetstone runs in %0.2f seconds. %0.2f whets/second\n",
   ticks/60.0, 60000000.0/ticks);
 getchar();
 exit(0);
```

```
pa(e)
double e[4];
 register int j;
 j = 0;
   lab:
 e[0] = (e[0] + e[1] + e[2] - e[3]) * t;
 e[1] = (e[0] + e[1] - e[2] + e[3]) * t;
 e[2] = (e[0] - e[1] + e[2] + e[3]) * t;
 e[3] = (-e[0] + e[1] + e[2] + e[3]) / t2;
 j += 1;
 if (j < 6)
   goto lab;
}
p3(x, y, z)
double x, y, *z;
 x = t * (x + y);
 y = t * (x + y);
 *z = (x + y) / t2;
p0()
 e1[j] = e1[k];
 e1[k] = e1[l];
 e1[1] = e1[j];
#ifdef POUT
pout(n, j, k, x1, x2, x3, x4)
intn, j, k;
double x1, x2, x3, x4;
 printf("%5d %5d %5d %11.3e %11.3e %11.3e %11.3e\n",
 n, j, k, x1, x2, x3, x4);
#endif
/*
 David Hinnant
  SCI Systems Inc.
  {decvax, akgua}!mcnc!rti-sel!scirtp!dfh
*/
```

USORT.PAS Program in Turbo Pascal 3.0 for the IBM PC and compatibles. From the article "Focus on Algorithms: Sorting out the Sorts" by Dick Pountain, July 1987, page 275.

```
program USORT;
const CR = #13; { carriage return character }
type letters = 'a' .. 'z';
  wordtype = string[16];
  nodeptr = 'nodetype;
  nodetype = record
info: wordtype;
next: nodeptr
end;
var inputFile,outputFile: text;
  inputFilename, outputFilename: string[127];
  chr, firstletter: char;
  sortList: array[letters] of nodeptr; { the array of 26 lists }
  i: letters;
  word: wordtype;
procedure InitFiles;
begin { open input and output files }
```

```
inputFilename := paramSTR(1);
 Assign(inputFile,inputFilename);
 Reset(inputFile);
 outputFilename := paramSTR(2);
 Assign(outputFile, outputFilename);
Rewrite(outputFile);
procedure GetWord(VAR infile: text; VAR word: wordtype);
begin { read a cleaned-up word from the input file }
 word := ''; { initialize to blank }
  read(infile,chr);
  if chr in ['A'..'Z'] { convert all to lowercase }
  then chr := char(ord(chr)+32);
  if chr in ['a'..'z'] { only accept alpha characters }
  then word := word+chr; { add to word being built }
 until (chr = ' ') or (chr = CR) or eof(infile)
procedure Place(VAR list: nodeptr; word: wordtype);
var p, q, newnode: nodeptr;
  found: boolean;
begin { insert new word into list in sorted position only if unique }
q := nil;
 p := list; { p points to head of list }
 found := false;
 while (p <> nil)
                  { not end of list and }
  and (not found) { word not already here and }
  and (word >= p^.info) do { word alphabetically later than current }
  if p^.info = word { does this node contain our word? }
  then found := true { yes! word is already here }
  else begin
  q := p; { remember this node and }
   p := p^.next { move on to the next one }
  end; {while}
 if not found { word isn't already here }
 then begin
  New(newnode); { create a new node }
  newnode . info := word; { put word in its info field }
  if q = nil { list was empty }
  then begin
  newnode .next := list; { newnode becomes first }
  list := newnode
  end
  else begin
  newnode .next := q .next; { insert after node q }
  q.next := newnode
  end
 end
end:
procedure SquirtOut(list: nodeptr; VAR outfile: text);
begin { send sorted list to output file }
 while list <> nil
 begin
  writeln(outfile, list'.info);
  list := list^.next
 end
end;
begin { main program }
 InitFiles;
 for i := 'a' to 'z' do sortList[i] := nil; { initialize all the lists }
 while not eof(inputFile) do
 begin
 GetWord(inputFile,word);
  firstletter := word[1]; { get first letter }
  Place(sortList[firstletter],word) { put word in proper place }
 end; {while}
 for i := 'a' to 'z' do SquirtOut(sortList[i],outputFile);
 writeln('Keywords are contained in ',outputFilename);
 Close(inputFile);
 Close(outputFile)
end.
```

{\$INCLUDE: 'complex.int'} IMPLEMENTATION OF complex; TYPE stackpt = 'stack; stack = RECORD r, i: REAL; {Holds real and imaginary parts of number} next, prev: stackpt; {Links RECORDs of stack} VAR zpt, zroot: stackpt; {variable stack pointer and root position} {Increments stack pointer; creates new RECORD only if next position = NIL} PROCEDURE push; VAR zsav: stackpt; IF (zpt^.next <> NIL) THEN zpt:= zpt^.next ELSE BEGIN zsav:= zpt; NEW(zpt); zpt^.prev:= zsav; zpt^.next:= NIL; zsav^.next:= zpt; END; END; PROCEDURE pop; {Decrements stack pointer} BEGIN IF (zpt^.prev <> NIL) THEN zpt:= zpt^.prev ELSE BEGIN {In case of no previous element, pop zeros} zpt^.r:= 0.0; zpt^.i:= 0.0; END; END; FUNCTION display: {Argument (indx: INTEGER) declared in interface; extracts real or imaginary parts of current stack pointee} BEGIN CASE indx OF 1: display:= zpt^.r; 2: display:= zpt^.i; OTHERWISE display: = 0; END; END; PROCEDURE keyin; {Argument (z: cmplx) declared in interface; equivalent of keying in numbers on calculator; pushes stack, inserts number at new pointee} BEGIN push; zpt^.r:= z[1]; zpt^.i:= z[2]; END; {Argument (x: REAL) declared in interface} PROCEDURE rkeyin; {Similar to KEYIN except enters real number} BEGIN push; zpt^.r:=x; zpt^.i:= 0.0; END; {Copies current pointee onto stack} PROCEDURE enter; VAR a, b: REAL; BEGIN a:= zpt^.r; b:= zpt^.i; push; zpt'.r:= a; zpt'.i:= b; END;

LISTING2.TXT Contributed by David Gedeon. Accompanies "Programming Insight: Complex Math in Pascal," July 1987, page 121.

```
PROCEDURE clear;
                     {Resets stack pointer to root of list, zeros}
       BEGIN
        zpt:= zroot;
        zpt^.r:= 0.0; zpt^.i:= 0.0;
        END;
 PROCEDURE negate; {Negative of current stack pointee}
        zpt^.r:=-zpt^.r;
        zpt^.i:= -zpt^.i;
        END:
 PROCEDURE conjugate; {Complex conjugate of current stack pointee}
        zpt^.i:= - zpt^.i;
        END;
 PROCEDURE invert;
                    {Inverse of current stack pointee}
       VAR mag: REAL;
       BEGIN
        mag:= (zpt^.r * zpt^.r) + (zpt^.i * zpt^.i);
        zpt^.r:= zpt^.r/mag;
        zpt^.i:=-zpt^.i/mag;
        END;
 PROCEDURE add;
                   {Adds current and previous stack pointees; pops stack; result in new pointee}
       VAR a, b: REAL;
       BEGIN
        a:= zpt^.r; b:= zpt^.i;
        pop:
        zpt^.r:= zpt^.r+a;
        zpt^.i:= zpt^.i+b;
        END;
PROCEDURE subtract;
                       {Subtracts current from previous stack pointee; pops stack; result in new pointee}
       BEGIN
       negate;
        add;
        END;
PROCEDURE multiply;
                      {Multiplies current and previous stack pointees; pops stack; result in new pointee}
      VAR a, b, c, d: REAL;
      BEGIN
       a:= zpt^.r; b:= zpt^.i;
       pop;
       c := (a * zpt^.r) - (b * zpt^.i);
       d:= (a * zpt^.i) + (b * zpt^.r);
       zpt^.r:= c;
       zpt^.i:=d;
       END:
PROCEDURE divide; {Divides previous stack pointee by current; pops stack; result in new pointee}
      BEGIN
       invert:
       multiply;
       END;
PROCEDURE cexp; {Complex exponential function of current stack pointee}
      VAR mag: REAL;
      BEGIN
       mag:= EXP(zpt^.r);
       zpt^.r:= mag*COS(zpt^.i);
       zpt^.i:= mag*SIN(zpt^.i);
       END;
PROCEDURE sinh; {Complex hyperbolic sine of current stack pointee}
      VAR z: cmplx;
      BEGIN
       z[1]:= zpt^.r; z[2]:= zpt^.i;
       cexp;
       keyin(z); negate; cexp; subtract;
       zpt^.r:= 0.5*zpt^.r; zpt^.i:=0.5*zpt^.i;
       END;
```

```
{Complex hyperbolic cosine of current stack pointee}
PROCEDURE cosh;
      VAR z: cmplx;
      BEGIN
       z[1]:= zpt^.r; z[2]:= zpt^.i;
       keyin(z); negate; cexp; add;
zpt^.r:= 0.5*zpt^.r; zpt^.i:= 0.5*zpt^.i;
        END;
            {Initialize stack pointer; define it as head; zero pointee}
BEGIN
     NEW(zpt);
     zroot:= zpt;
     zpt^.prev:= NIL; zpt^.next:= NIL;
     clear;
END.
```

LISTING.TXT Contributed by David Gedeon. Accompanies "Programming Insight: Complex Math in Pascal," July 1987, page 121.

```
LISTING 1 - available operations
 PROCEDURE negate;
{Negative of current stack pointee}
 BEGIN
  zpt^.r:=-zpt^.r;
  zpt^.i := -zpt^.i;
 END:
 PROCEDURE conjugate;
{Complex conjugate of current stack pointee}
 BEGIN
  zpt^.i := - zpt^.i;
 END;
 PROCEDURE invert;
{Inverse of current stack pointee}
  VAR mag: REAL;
  mag := (zpt^.r * zpt^.r) + (zpt^.i * zpt^.i);
   zpt^.r:= zpt^.r/mag;
  zpt^.i := -zpt^.i/mag;
  END;
  PROCEDURE add;
{Adds current and previous stack pointees; pops stack; result in new pointee}
  VAR a, b: REAL;
  BEGIN
   a := zpt^.r; b := zpt^.i;
   pop;
   zpt^.r:= zpt^.r+a;
   zpt^.i := zpt^.i + b;
  END;
  PROCEDURE subtract;
 {Subtracts current from previous stack pointee; pops stack; result in new pointee}
  BEGIN
   negate;
   add;
  END;
  PROCEDURE multiply;
 {Multiplies current and previous stack pointees; pops stack; result in new pointee}
  VAR a, b, c, d: REAL;
  BEGIN
   a := zpt^.r; b := zpt^.i;
   c := (a * zpt^.r) - (b * zpt^.i);
   d := (a * zpt^.i) + (b * zpt^.r);
```

July

```
zpt^.r:=c;
  zpt^.i := d;
 END;
 PROCEDURE divide;
{Divides previous stack pointee by current; pops stack; result in new pointee}
 BEGIN
  invert;
  multiply;
 END;
 PROCEDURE cexp;
{Complex exponential function of current stack pointee}
 VAR mag: REAL;
 BEGIN
  mag := EXP(zpt^.r);
  zpt^.r := mag*COS(zpt^.i);
  zpt^.i := mag*SIN(zpt^.i);
 END;
 PROCEDURE sinh;
{Complex hyperbolic sine of current stack pointee}
 VAR z: cmplx;
 BEGIN
 z[1] := zpt^.r; z[2] := zpt^.i;
 keyin(z); negate; cexp; subtract;
  zpt^.r:=0.5*zpt^.r; zpt^.i:=0.5*zpt^.i;
 END;
PROCEDURE cosh;
{Complex hyperbolic cosine of current stack pointee}
VAR z: cmplx;
BEGIN
 z[1] := zpt^.r; z[2] := zpt^.i;
 cexp;
 keyin(z); negate; cexp; add;
 zpt^.r:=0.5*zpt^.r; zpt^.i:=0.5*zpt^.i;
END:
```

QDRCNINT.CRV Contributed by William A. McWorter Jr.
Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
4
          8
    2
    OR
TA
     5
          1
2
    90R
TA
     2
2
          1
    180R
TA
2
     7
          3
    270R
TA
2
     0
          3
TA
    ONBR
2
     0
TA
    90NBR
     5
          6
2
TA
    180NBR
2
     2
          6
TA
    270NBR
 2
     7
          4
 1
      1
          75
               140
 6
      13
```

QDEKNGCH.CRV Contributed by William A. McWorter Jr.
Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
4
    4
          4
TA
    OR
               0
                    3
4
     0
          1
    90R
TA
          3
               2
                    3
4
     2
    180R
TA
                    0
2
    2
          1
               0
    270R
TA
               0
                    0
2
     0
          1
4
     0
          3
          100
                    140
     7
```

QHILBRT.CRV Contributed by William A. McWorter Jr. Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
4
     4
           10
     OR
TA
4
           0
                3
                     4
     1
TA
     90R
                2
                     5
 4
      0
           1
TA
     180R
                2
 4
     0
           1
TA
     270R
 4
                3
                     7
     1
           0
TA
     OR
 4
      6
                4
                     0
TA
     90R
      7
           6
                 5
                      1
```

TA	180R			
4	7	6	5	2
TA	270R			
4	6	7	4	3
TA	OR			
4	1	0	3	9
TA	90R			
4	6	7	4	8
1	8			
10	5	80		190



QSRPNSK.CRV Contributed by William A. McWorter Jr.

Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

4	9	8							
TA	OR								
9	0	1	0	3	6	3	0	1	0
TA	90R								
9	1	2	1	0	7	0	1	2	1
TA	180R								
9	2	3	2	1	4	1	2	3	2
TA	270R								
9	3	0	3	2	5	2	3	0	3
TA	OBR								
3	4	4	4	0	0	0	0	0	0
TA	90BR								
3	5	5	5	0	0	0	0	0	0
TA	180BR								
3	6	6	6	0	0	0	0	0	0
TA	270BR								
3	7	7	7	0	0	0	0	0	0
1	0								
5	4	50	100						

QPENTGRE.CRV Contributed by William A. McWorter Jr. Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
6
          5
TA
     OR
                    2
                             0
                         1
TA
    72R
     1
          0
               2
                        2
                    3
                             1
TA
    144R
6
     2
               3
                        3
                             2
TA
    216R
 6
     3
                    0
                        4
                             3
TA
    288R
6
     4
                    1
                        0
1
     0
          500
                    150
```

QSNOFLK.CRV Contributed by William A. McWorter Jr.

Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
6 4 6
TA OR
4 0 1 5 0
TA 60R
4 1 2 0 1
```

```
120R
TA
 4
     2
                 1
                      2
           3
TA
     180R
 4
                 2
                      3
      3
     240R
TA
 4
      4
                 3
                      4
     300R
TA
 4
      5
                      5
 1
      0
 2
      5
            50
                   150
```

QAROHEAD.CRV Contributed by William A. McWorter Jr.
Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
24
6
     3
     OR
TA
     0
           7
                17
3
TA
     60R
3
           8
                12
     1
TA
     120R
3
     2
                13
     180R
TA
     3
240R
           10
                14
3
TA
     4
           11
                15
3
TA
     300R
3
     5
           6
                16
     OR
TA
3
     23
                6
           1
TA
     60R
                 7
3
      18
           2
     120R
TA
 3
      19
           3
                 8
     180R
TA
3
      20
                 9
     240R
TA
.3
      21
                 10
TA
     300R
 3
      22
           0
                 11
TA
     OR
 3
      12
            23
                 1
TA
     60R
 3
      13
            18
                 2
TA
     120R
 3
      14
            19
                 3
TA
     180R
                 4
 3
      15
            20
TA
     240R
 3
      16
            21
                 5
TA
     300R
 3
      17
            22
                 0
TA
     OR
 3
      7
            17
                 18
TA
     60R
 3
            12
                 19
TA
     120R
 3
      9
            13
                 20
     180R
TA
 3
      10
            14
                 21
TA
     240R
 3
      11
            15
                 22
TA
     300R
      6
            16
                 23
 344
      2
                 17
            7
                      22
            320
                   195
```

QBRKINT.CRV Contributed by William A. McWorter Jr.
Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
2
           8
TA
     ONBR
 2
      0
           1
TA
     90NBR
 2
           6
TA
    180NBR
2
     2
           3
TA
    270NBR
2
     0
TA
    OR
2
TA
    90R
2
           6
TA
    180R
2
          6
TA
    270R
2
     0
1
     0
8
     12
          225 150
```

QBRICK.CRV Contributed by William A. McWorter Jr.
Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
4
      2
           6
TA
     OR
 2
TA
     90R
 2
           5
TA
     180R
 2
      2
           3
TA
     270R
 2
      0
TA
     OR
 2
     1
     180R
TA
2
     3
2
     0
8
      10
           225 100
```

QLACE.CRV Contributed by William A. McWorter Jr.

Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
12
            24
TA
     60R
 3
      0
           1
                 2
TA
     120R
 3
      3
TA
     30R
           7
 3
      6
                0
     300R
TA
 3
                9
TA
     OR
 3
     10
           11
                12
TA
     330R
 3
      7
           0
                1
TA
     300R
3
      8
           13
                14
TA
     OR
3
     15
           10
                16
```

```
180R
TA
 3
      6
           7
                17
     150R
TA
 3
     11
           18
                3
TA
     120R
           15
                19
 3
     180R
TA
 3
      13
           6
                20
TA
     90R
                4
 3
      18
           3
     240R
TA
 3
      11
           18
                21
     150R
TA
      8
           13
                6
     60R
TA
 3
      7
           0
                22
TA
     90R
                7
           6
 3
      13
     270R
TA
      0
           1
                8
 3
TA
     240R
 3
      18
           3
                 23
TA
     330R
      4
                10
 3
           15
TA
     270R
 3
      15
           10
                11
TA
     210R
           8
                 13
 3
      1
TA
     30R
      3
           4
                 15
 3
     210R
TA
                 18
 3
      10
           11
 1
      0
 10
      6
            10
                 180
```

QDRGBDRY.CRV Contributed by William A. McWorter Jr. Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
4
      2
           12
     OR
TA
 2
      0
     90R
TA
 2
      6
           9
TA
     180R
           2
 2
      7
     270R
TA
           4
 2
      11
TA
     OR
 1
      8
           0
TA
     90R
 1
      6
           0
TA
     180R
           0
      10
 1
TA
     270R
      4
           0
 1
TA
     OR
 2
      0
            1
TA
     90R
 2
           9
      10
TA
     180R
            2
 2
      3
TA
     270R
 2
            8
      11
      0
                6
                     11
 3
      13
             200
                     80
```

QMDLQUIN.CRV Contributed by William A. McWorter Jr.
Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
4
     5
          8
TA
    OR
         0
                  6
 5
    4
              3
                       3
TA
    90R
              0
                  7
                       0
 5
     5
    180R
TA
 5
    6
         2
              1
                       1
    270R
TA
    7
              2
                       2
5
                  5
TA
    OR
              7
5
    7
         2
                  4
                       0
    90R
TA
5
    4
         3
                  5
                       1
TA
    180R
                       2
5
    5
         0
              5
                  6
TA
    270R
                  7
5
     6
              6
                       3
1
     0
 5
     5
         300
                  190
```

QMOORE.CRV Contributed by William A. McWorter Jr.

Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

4 9 8 TA OR 9 0 1 2 6 2 3 7 6 5 TA 90R 9 1 2 3 7 3 0 4 7 6 TA 180R 9 2 3 0 4 0 1 5 4 7 TA 270R 9 3 0 1 5 1 2 6 5 4 TA OR										
9 0 1 2 6 2 3 7 6 5 TA 90R 9 1 2 3 7 3 0 4 7 6 TA 180R 9 2 3 0 4 0 1 5 4 7 TA 270R 9 3 0 1 5 1 2 6 5 4	4	9	8							
TA 90R 9 1 2 3 7 3 0 4 7 6 TA 180R 9 2 3 0 4 0 1 5 4 7 TA 270R 9 3 0 1 5 1 2 6 5 4	TA	OR								
9 1 2 3 7 3 0 4 7 6 TA 180R 9 2 3 0 4 0 1 5 4 7 TA 270R 9 3 0 1 5 1 2 6 5 4	9	0	1	2	6	2	3	7	6	5
TA 180R 9 2 3 0 4 0 1 5 4 7 TA 270R 9 3 0 1 5 1 2 6 5 4		90R								
9 2 3 0 4 0 1 5 4 7 TA 270R 9 3 0 1 5 1 2 6 5 4			2	3	7	3	0	4	7	6
TA 270R 9 3 0 1 5 1 2 6 5 4										
9 3 0 1 5 1 2 6 5 4			3	0	4	0	1	5	4	7
TA OR			0	1	5	1	2	6	5	4
9 1 2 3 7 6 2 6 5 4			2	3	7	6	2	6	5	4
TA 90R										
9 2 3 0 4 7 3 7 6 5			3	0	4	7	3	7	6	5
TA 180R										
9 3 0 1 5 4 0 4 7 6			0	1	5	4	0	4	7	6
TA 270R										
9 0 1 2 6 5 1 5 4 7			1	2	6	5	1	5	4	7
1 0										
4 4 200 100	4	4	200		100					

QSRPNSK2.CRV Contributed by William A. McWorter Jr. Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
8
     2
          8
TA
    OR
 2
     0
          1
    315R
TA
 2
          0
TA
    270R
 2
TA
    90R
 2
     2
TA
    45R
 2
     6
TA
    225R
 2
     0
          3
```

```
TA
    180R
2
     6
TA
    135R
2
     3
           6
     4
1
5
           580 150
      11
```

QCHRSTRE.CRV Contributed by William A. McWorter Jr. Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
3
           10
 5
TA
     OR
                6
           0
 3
     9
     72R
TA
      5
           1
                7
 3
TA
     144R
3
     6
           2
                 8
TA
     216R
                9
3
     7
           3
TA
     288R
           4
                 5
3
     8
TA
     OR
           5
                 4
 3
      1
TA
     72R
 3
     2
           6
                 0
TA
     144R
           7
 3
     3
                 1
TA
     216R
                 2
 3
     4
           8
TA
     288R
           9
                 3
 3
      0
 1
 10
      7
            180
                   190
```

QDRGNCRD.CRV Contributed by William A. McWorter Jr. Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
4
      2
           8
TA
     OR
 2
      0
           5
TA
     90R
 2
      2
     180R
TA
 2
      2
           7
     270R
TA
 2
      0
     OBR
TA
 2
      4
     90BR
TA
 2
      6
     180BR
TA
 2
      6
           3
     270BR
TA
 2
      4
           3
      0
 1
      11
           400
                   150
```

```
QDRGN.CRV Contributed by William A. McWorter Jr.
Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.
```

```
4
     2
         4
TA
    OR
2
     0
TA
    90R
2
    2
TA
    180R
2
    2
    270R
TA
2
    0
         3
1
    0
8
    11 400 150
```

QGOSPER.CRV Contributed by William A. McWorter Jr.
Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
6
     7
         12
TA
    OR
          7
     0
              9
                  2
                       0
                           0
                               11
TA
    60R
     1
          8
              10 3
                       1
                           1
                               6
TA
    120R
7
     2
              11
                  4
                       2
                           2
                               7
TA
    180R
7
    3
         10
              6
                  5
                       3
                           3
                               8
    240R
TA
7
         11 7
                  0
                       4
                           4
                               9
TA
    300R
7
              8
                  1
                       5
                           5
                               10
TA
    OR
7
    5
              6
         6
                  8
                      3
                               6
                           1
TA
    60R
7
     0
                  9
                           2
                               7
TA
    120R
7
         8 8 10
                      5
                           3
                               8
TA
    180R
7
    2
         9
                  11
                      0
                           4
                               9
TA
    240R
7
    3
         10
             10 6
                      1
                           5
                              10
TA
    300R
7
     4
         11 11
                           0
                              11
3
     0
              2
10
    3
         220
               120
```

SZPAK.LST The following five listings accompany "Logic Grammars" by Stan Szpakowicz, August 1987, page 185.

```
Listing 1:
statements --> statement, [';'], statements.
statements --> [].
statement --> [skip].
statement --> [id(V)], [:=], expr.
statement --> [if], condition, [then], statements, [fi].
statement --> [while], condition, [do], statements, [od].
condition --> [not], relation.
condition --> relation.
relation --> expr, comp_op, expr.
comp_op --> ['='].
comp_op --> ['<'].
```

```
expression --> primary.
expression --> expression, arith_op, primary.
primary --> [id(V)].
primary --> [num(N)].
arith_op --> ['+'].
arith_op --> ['-'].
arith_op --> ['*'].
arith_op --> ['/'].
[end listing 1]
Listing 2:
/*1*/ statement(K, N) :-
       token(id(V), K, L), token(:=, L, M), expr(M, N).
/*2*/ expr(K, L) :- primary(K, L).
/*3*/ \exp(K, N) := \exp(K, L), \operatorname{arith_op}(L, M), \operatorname{primary}(M, N).
/*4*/ primary(K, L) :- token(id(V), K, L).
/*5*/ primary(K, L) :- token(num(V), K, L).
/*6*/ arith_op(K, L):- token(+, K, L).
/*7*/ token(T, [T|Ts], Ts).
[end listing 2]
Listing 3:
program(s(Stmt, Stmts)) -->
        statement(Stmt), [';'],
        statements(Stmts).
statements(s(Stmt, Stmts)) -->
        statement(Stmt), [';'],
        statements(Stmts).
statements(skip) --> [].
% a sequence of statements is represented as a nested term,
% for example s(Stmt1, s(Stmt2, s(Stmt3, skip))),
% where Stmt1, Stmt2, Stmt3 represent individual statements
statement(skip) --> [skip].
statement(let(V, E)) \longrightarrow [id(V)], [:=], expr(E).
statement(if(C, Stmts)) -->
        [if], condition(C), [then], statements(Stmts), [fi].
statement(while(C, Stmts)) -->
        [while], condition(C), [do], statements(Stmts), [od].
condition(not(C)) --> [not], relation(C).
condition(C) --> relation(C).
relation(cond(Op, E1, E2)) --> expr(E1), comp_op(Op), expr(E2).
comp_op('=') --> ['='].
comp_op('<') --> ['<'].
[end listing 3]
Listing 4:
interm_code(s(Stmt, Stmts)) -->
        interm_code(Stmt), interm_code(Stmts).
interm_code(skip) --> [].
interm_code(let(V, E)) -->
        expr_interm_code(E), [store(V)].
interm_code(if(C, Stmts)) -->
        { newlabel(L) },
        cond_interm_code(not(C)),
        [jmp_cond(L)],
        interm_code(Stmts),
        [label(L)].
interm_code(while(C, Stmts)) -->
         { newlabel(L1) }, { newlabel(L2) },
        [label(L1)],
        cond_interm_code(not(C)),
         [jmp_cond(L2)],
        interm_code(Stmts),
        [jmp(L1)], [label(L2)].
[end listing 4]
```

```
Listing 5:
The source program:
x:=a; y:=n; z:=1;
while not i < 1 do
  if y/2*2< y then
   z := z * x;
  fi;
  x := x * x;
  y:=y/2;
od; #
The resulting object code:
load(a)
store(x)
load(n)
store(y)
loadc(1)store(z)
label($1b11)
loadc(1)
store($mem1)
load(i)
sub($mem1)
tst_neg
jmp_cond($1b12)
[end listing 5]
```

SZPAK.BNL Contributed by Stan Szpakowicz.

Accompanies "Logic Grammars" by Stan Szpakowicz, August 1987, page 185. Written in Prolog using logic grammars.

```
%
% Note: In order to execute this program, a Prolog interpreter must support logic grammars or definite-clause grammars
% === main program ===
compile :-
        set_gensym("$1b1"), set_gensym("$mem"),
        read_in(Chars),
                                                              % (skips initial white space)
        lsym_list(LexSyms, Chars, []),
                                                              % lexical analysis
        program( Tree, LexSyms, []),
                                                              % syntactic analysis
        interm_code( Tree, Code, []),
                                                              % code generation
        write_out(Code), !.
compile :- write( 'Sorry'), nl.
% read in a sequence of characters terminated by a #
read_in(Chars):-get(Ch), read_in(Ch, Chars).
read_in(35,[]):-!.
read_in(Ch, [Ch | Chars]):-getO(Ch1), read_in(Ch1, Chars).
% print the generated code one instruction per line
write_out([]).
write_out([Instr | Instrs]):-
        write(Instr), nl, write_out(Instrs).
% === scanner ===
% list of lexical symbols
lsym_list([LexSym | LexSyms]) -->
        lsym(LexSym),!, opt_space, lsym_list(LexSyms).
lsym_list([]) -->[].
% one lexical symbol (input tokens are ASCII codes)
lsym(IdOrKwd) --> letter(L), alphanums(Ls),
                  { name(Nm, [L | Ls] ) }, { wrap_name(Nm, IdOrKwd) }.
lsym( num( N ) ) --> digit( D ), digits( Ds ),
                  { name( N, [D | Ds] ) }.
lsym(:=)-->[58],[61].
                                       % colon, equals
lsym(S) --> [Ch], { name(S, [Ch]) }.
```

```
% optional white space between lexical symbols
opt_space --> white_space, !, opt_space.
opt_space --> [].
% recognizing classes of ASCII codes
letter(L) --> [L], { is_letter(L) }.
digit(D) --> [D], { is_digit(D) }.
white_space --> [Ch], { is_white_space(Ch) }.
is_letter(Ch):-65 = \langle Ch, Ch = \langle 90.
is_letter(Ch):-97 = Ch, Ch = 122.
is_digit(Ch):-48 = < Ch, Ch = < 57.
is_white_space(32).
                                  % blank space
is_white_space(13).
                                  % new line (this would be 10 in Quintus Prolog)
is_white_space(9).
                                  % tab
% keywords and identifiers
alphanums([L \mid Ls]) --> letter(L), alphanums(Ls). alphanums([L \mid Ls]) --> digit(L), alphanums(Ls).
alphanums([]) -->[].
wrap_name(Nm, Nm):-is_keyword(Nm).
wrap_name(Nm, id(Nm)).
% table of keywords
                                                           is_keyword(fi).
                              is_keyword( then ).
is_keyword(if).
is_keyword( while ).
                            is_keyword(do).
                                                           is_keyword(od).
is_keyword(skip).
                             is_keyword( not ).
% integers
digits([D | Ds]) --> digit(D), digits(Ds).
digits([]) --> [].
% === parser ===
program(s(Stmt, Stmts)) -->
         statement(Stmt), [';'],
        statements (Stmts).
statements(s(Stmt, Stmts)) -->
         statement(Stmt), [';'],!,
         statements (Stmts).
statements(skip) --> [].
% a sequence of statements is represented as a nested term,
% for example s(Stmt1, s(Stmt2, s(Stmt3, skip))),
% where Stmt1, Stmt2, Stmt3 represent individual statements
statement(skip) --> [skip].
statement( let( V, E ) ) --> [id( V )], [:=], expr( E ).
statement(if(C, Stmts)) -->
         [if], condition(C), [then], statements(Stmts), [fi].
statement(while(C, Stmts)) -->
         [while], condition(C), [do], statements(Stmts), [od].
condition(not(C)) --> [not], relation(C).
condition(C) --> relation(C).
relation(cond(Op, E1, E2)) --> expr(E1), comp_op(Op), expr(E2).
comp_op('=') --> ['='].
comp_op('<') --> ['<'].
expr(E) --> add_expr(AE), rest_expr(AE, E).
rest_expr(AE1, E) -->
         ['+'], add_expr(AE2), rest_expr(e('+', AE1, AE2), E).
rest_expr(AE1, E) -->
         ['-'], add_expr(AE2), rest_expr(e('-', AE1, AE2), E).
rest_expr(E, E) -->[].
add_expr( AE ) --> mult_expr( ME ), rest_add_expr( ME, AE ).
rest_add_expr( ME1, AE ) -->
         ['*'], mult_expr( ME2 ), rest_add_expr( e( '*', ME1, ME2 ), AE ).
```

```
rest_add_expr( ME1, AE ) -->
         ['/'], mult_expr( ME2 ), rest_add_expr( e( '/', ME1, ME2 ), AE ).
 rest_add_expr(E, E) --> [].
 mult_expr(var(V)) --> [id(V)].
 mult_expr(num(N)) --> [num(N)].
 mult_expr(E) --> ['('], expr(E), [')'].
 % === code generation ===
 % statements
 interm_code(s(Stmt, Stmts)) -->
         interm_code(Stmt), interm_code(Stmts).
 interm_code(skip) --> [].
 interm_code(let(V, E)) -->
         expr_interm_code(E), [store(V)].
 interm_code(if(C, Stmts)) -->
         { newlabel(L) },
         cond_interm_code(not(C)),
         [jmp_cond(L)],
         interm_code(Stmts),
         [label(L)].
 interm_code(while(C, Stmts)) -->
         { newlabel(L1)}, { newlabel(L2)},
         [label(L1)],
         cond_interm_code(not(C)),
         [jmp_cond(L2)],
         interm_code(Stmts),
         [jmp(L1)], [label(L2)].
% conditions
cond_interm_code( not( not( C ) ) ) --> cond_interm_code( C ).
cond_interm_code(not(R)) -->
        rel_interm_code(R),[flip].
                % flip: negate the contents of the condition register
cond_interm_code(R) -->
        rel_interm_code(R).
% relations
rel_interm_code(cond(Op, E1, E2)) -->
        expr_interm_code(E2), { newmemloc(M)}, [store(M)],
        expr_interm_code(E1), [sub(M)], tst_interm_code(Op).
% set the condition register
tst_interm_code( '=') --> [tst_zer].
tst_interm_code('<') --> [tst_neg].
% expressions
expr_interm_code(e(Op, E1, E2)) -->
        expr_interm_code(E2), { newmemloc(M)}, [store(M)],
        expr_interm_code(E1), eop_interm_code(Op, M).
expr_interm_code( var( V ) ) -->
        [load(V)].
% load a constant
expr_interm_code( num( N ) ) -->
        [loadc(N)].
eop_interm_code('+', M) --> [add(M)].
eop_interm_code('-', M) --> [sub(M)].
eop_interm_code('*', M) --> [mul(M)].
eop_interm_code('/', M) --> [div(M)].
% auxiliaries
newlabel( L ) :-
        gensym( "$1b1", L).
newmemloc(M):-
       gensym("$mem", M).
% === utilities ===
% symbol generator (preset in the main program)
set_gensym( Pref ) :-
       retract( sym( Pref, _ ) ), fail.
set_gensym( Pref ) :-
       assert(sym(Pref, 1)).
gensym( Pref, Sym ) :-
       retract( sym( Pref, Num ) ),
```

```
Num1 is Num + 1,
    assert(sym(Pref, Num1)),
    glue(Pref, Num, Sym).

glue(Pref, Num, Sym):-
    name(Num, Digits), append(Pref, Digits, All),
    name(Sym, All),!.

% well, you can't have a program without append...
append([], Z, Z).
append([A | X], Y, [A | Z]):- append(X, Y, Z).

% end of program
```

DRAGON.BAS Contributed by William A. McWorter Jr.

Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

```
DRAGON PROGRAM
3 'This is the DRAGON. BAS program in Microsoft BASIC for the IBM PC and compatibles. If you have QuickBASIC, QDRAGON. BAS is
 4 'easier to use and comes with quite a few dragon files (.CRV) that will make your fractal wanderings considerably easier.
5 'If you do not have that package, this program is fun and allows you to explore the dragons. It is not, however,
6 'forgiving of mistakes. Any data-entry mistakes require that you start over. Have a good time!
71
8 1
91
11 '
12 'To avoid local variables, I have replaced the loop index K with a one-dimensional array K, which holds the loop index
13 'at each level of recursion, and exchanged the variable CELL for a one-dimensional array CELL, which holds the cell
14 'value at each level of recursion. Since Microsoft BASIC does not permit subscripted variables to be FOR...NEXT loop
15 'indexes, I have replaced this loop with a WHILE...WEND block.
16 1
17 '
18 '
20 '----Initialize
30 DATA 14,12,11,13,10,9,15
40 CLEAR,,10000
42 CLS
.RM63/46 FOR I=1 TO 7
48 READ X
49 NEXT T
50 '----Get dragon
60 INPUT; "DRAGON FROM DISK (<CR> if no) ",C$:IF C$>"" THEN GOSUB 360:PRINT:GOTO 210
70 INPUT"NUMBER OF DIRECTIONS"; D: INPUT"NUMBER OF CELLS (equal or greater the number of directions)"; M: IF M<D THEN 70
80 INPUT"MAX NUMBER OF CELLS IN A CELL DIVISION"; L: PRINT"Follow each input cell with <CR>":DIM G(M-1,L)
90 PRINT"cyclic? (<CR> if no)": IF INPUT$(1)=CHR$(13) THEN 140
100 J=0:WHILE J<M:PRINT"division of cell"J"(enter '.' after last cell of division less than"L:G(J,0)=L
110 FOR I=1 TO L:INPUT; " ",A$:G(J,I)=VAL(A$):IF INSTR(A$,".") THEN G(J,O)=I:I=L
120 NEXT I:PRINT:FOR I=1 TO G(J,0):S=G(J,I):T=S-(S MOD D):FOR K=J+1 TO J+D-1:G(K,0)=G(J,0):G(K,I)=((S+K) MOD D)+T:NEXT K,I
130 J=J+D:WEND:GOTO 170
140 FOR J=0 TO M-1: PRINT"DIVISION OF CELL"J" (enter (.) after last cell of divisions less than"L:G(J,0)=L
150 FOR I=1 TO L:INPUT; "", A$:G(J,I)=VAL(A$):IF INSTR(A$,".") THEN G(J,O)=I:I=L
160 NEXT I:PRINT:NEXT J
170 DIM I(M-1):PRINT"If cell directions are CELL modulo NUMBER OF DIRECTIONS, then press <CR>":IF INPUT$(1)=CHR$(13) THEN FOR I=O
TO M-1:I(I)=I MOD D:NEXT I:GOTO 210
180 FOR I=O TO M-1:PRINT"DIRECTION FOR CELL"I;:INPUT I$:IF I$="." THEN I(I)=-1 ELSE I(I)=VAL(I$)
190 NEXT I
200 '-----Compute direction vectors
210 T=6.28318531#/D:DIM X(D-1),XX(D-1),Y(D-1),YY(D-1):FOR I=0 TO D-1:XX(I)=COS(I*T):YY(I)=SIN(I*T):NEXT I
220 '----Get drawing parameters
230 INPUT"NUMBER OF BIRTH CELLS IN START PATTERN "; T: PRINT"Follow each birth cell input with <CR>":DIM W(T-1): FOR I=0 TO
T-1: INPUT; " ", W(I): NEXT I: PRINT
240 INPUT; "AGE IN DAYS ", DAY: INPUT; " CELL LENGTH ", W: FOR I = O TO D-1: X(I) = W*XX(I): Y(I) = W*YY(I): NEXT I: PRINT
250 INPUT; "COORDINATES OF HEAD CELL (", X: INPUT; ", ", Y: PRINT") ": DIM K(DAY), CELL(DAY)
260 '----Draw dragon and repeat
270 CLS
271 SCREEN 2
273 PRESET(X,Y)
275 FOR I=0 TO T-1
277 CELL(DAY)=W(I)
```

```
278 GOSUB 310
279 NEXT I
280 PRINT"(1) NEW BIRTH CELL LIST, (2) NEW AGE, (3) SAVE TO DISK ": A$=INPUT$(1): ON INSTR("123", A$) GOTO 330,340,400
290 RIIN
          --- Dragon procedure
310 IF DAY=0 THEN CODE=I(CELL(DAY)):GOSUB 450:RETURN
320 K(DAY)=1:WHILE K(DAY)<=G(CELL(DAY),0):CELL(DAY-1)=G(CELL(DAY),K(DAY)):DAY=DAY-1:GOSUB
310:DAY=DAY+1:K(DAY)=K(DAY)+1:WEND:RETURN
330 ERASE W, K, CELL: GOTO 230
340 ERASE K, CELL: GOTO 240
350 '----Load dragon routine
360 OPEN"I",1,C$+".CRV":INPUT#1,D,M,L:DIMG(M-1,L),I(M-1)
370 FOR I=O TO M-1: INPUT# 1,G(I,O): FOR J=1 TO G(I,O): INPUT# 1,G(I,J): NEXT J,I
380 FOR I=O TO M-1: INPUT# 1, I(I): NEXT I: CLOSE: RETURN
390 '----Save dragon routine
400 INPUT"CURVE NAME"; C$: OPEN "O", 1, C$+". CRV"
410 PRINT# 1,D;M;L:FOR I=O TO M-1:PRINT# 1,G(I,O);:FOR J=1 TO G(I,O)
420 PRINT# 1,G(I,J);:NEXT J:NEXT I:FOR I=O TO M-1:PRINT# 1,I(I);:NEXT I
430 CLOSE: GOTO 280
440 '----Routine to interpret a cell
450 IF CODE<0 THEN RETURN
460 IF CODE<D THEN X=X+X(CODE):Y=Y-Y(CODE):LINE-(X,Y), I MOD 7+1:RETURN
470 X=X+X(CODE-D):Y=Y-Y(CODE-D):PRESET(X,Y):RETURN
```

QRULES.TXT Contributed by William A. McWorter Jr. Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

Running QDRAGON

The new QDRAGON. BAS is written in QuickBASIC to exploit the DRAW commands and to make data entry less tedious. As I understand it, only the SCREEN commands must be changed to suit the graphics adapter available. The LINE commands have been replaced by DRAW commands to permit drawing cells that are more exotic than simple line segments. The default is a line segment drawn in a direction determined by the cell label number. The length of the line segment is determined by scale factor held in the "cell length" variable. If something else is desired, you must type in the appropriate sequence of DRAW commands. Such a sequence will be sized by the value in the "cell length" variable. The program displays the DRAW commands it will use.

QDRAGON.BAS improves on the user interface but is not yet ideal. It pretends it knows what you want as data and displays what it will use if you don't enter any data. Pressing Return repeatedly will show currently recorded data and how the program wants the data typed in. Pressing F1 at any time will draw the dragon determined by current data. The program always has complete data to draw some dragon. To enter data for a new dragon, simply enter the data requested followed by Return. To back up and change or view previously requested data, end entry with a].

The first thing the program wants to know is if you want a dragon stored on disk. Pressing Return or] ignores this request and displays the next, or previous, thing it wants. Otherwise you type in the name of a dragon stored on disk followed by Return or], or F1 if you want the program to draw the dragon you have named. Dragons stored on disk contain *all* the data required by the program, including positioning on the screen and the birth cell list. Hence loading a dragon from disk can be followed immediately by F1 which will draw it on the screen.

The next thing the program wants is the number of directions. This request is the number of cell orientations. Entering the number 4 means all cells are oriented in four directions, east, north, west, or south, as if reading a map. The program assumes that all cells are single line segments oriented in one of four directions unless told otherwise by you or the input file. In any case, the program will display the DRAW commands it will use. Entering the DRAW commands wanted (followed by Return,], or F1) changes the default value. Pressing] following data entry displays what the program thinks you typed.

Following the 'number of directions' request, QDRAGON.BAS wants the maximum cell division. This is so that the program can begin to dimension the genetic code array. You type in this value if it differs from the displayed value (Return cycles forward through data entry). The genetic code requires two dimensions, so if F1 is pressed now, all entries in the genetic code array will be zero, producing a dull dragon. The program redimensions the genetic code array whenever one of its dimensions is changed.

Next, the program wants the total number of cell types. This number completes the dimension of the genetic code array. The program displays, as always, what it thinks should be the number of cell types. Typing Return accepts the program's choice and displays what the program wants next and the value it will use if you don't change it.

After typing Return, the program displays the DRAW commands it will execute for the first cell. You should type Return to go on to the next program request, or type in the DRAW commands you want for the first cell followed by Return to advance to the next request or] to edit or preview the previous request. If you want a cell to be an invisible line segment, typing a dash will cause the program to record the appropriate DRAW commands, and if you want a 'do nothing' cell, typing * makes the program fill in DRAW commands that do the job.

Ending the entry with Return, the program will now exhibit the label of the first child of the first cell. Each cell divides into a number of cells, numbered from one to the number of cells into which the cell divides. A user entry here should be a cell label, a number between 0 and one less than the total number of cells. Typing Return causes the program to ask for the next child of the current cell. If the cell has fewer children than the maximum, the last child cell should be followed by a period.

Whenever the program asks you for the last child of a cell labelled a multiple of the number of directions, it permits you to end entry with an * to abort this feature. Without the * entry, the program will fill in child cells cyclically up to the next cell labelled a multiple of the number of directions and display the draw code for the next cell, which is a multiple of the number of directions.

After all cells and their children have been displayed, the program requests the number of cells in the birth cell list. Dragons can begin life as a single cell (number of cells in birth cell list, 1) or as several cells, each drawn in a different color. Next the dragon's age in days is requested. Following this comes a request for cell length and the position on the screen where drawing is to begin. The program draws the dragon starting at this position.

Finally, the program asks if the dragon just drawn is to be saved on disk. If so, you type a name of up to 8 letters followed by Return or]. Only Control-Break or illegal data entry ends the program.

QDRAGON. BAS displays DRAW commands instead of numbers, so ignore interpreter values in the table. The relation between interpreter values in the table and the draw codes in the program is I->TA360I/DR. Invisible cells end in BR and 'do nothing' cells end in NBR.

Have a good time exploring fractals. I have.

-William A. McWorter Jr.

t=pos(0)+1:print 1;:gosub getcmd

if a\$=chr\$(0) then goto program if a\$="]" then goto drctns

if i\$>" "then l=val(i\$):redim g(m-1,1),i\$(m-1)

QDRAGON.BAS Contributed by William A. McWorter Jr.
Accompanies "Programming Project: Creating Fractals" by William A. McWorter Jr. and Jane Morrill Tazelaar, August 1987, page 123.

d=4:1=2:m=4:w=1:s=10:n=5:x=320:y=175 $\dim i\$(m-1),g(m,1),w(w):a=360/d$ print"The program tries to anticipate your wishes." print"You can cycle through the data for previewing" print"or editing, or enter entirely new data." print" <cr> cycles forward through data entry" print"<]> cycles backwards through data entry." print"<F1> at any time draws the dragon with current data." print"But pressing it now will get you only an empty screen." print"The backspace key allows erasing the last-typed character." print begin:print"input dragon? (<cr> or <]> if no) ";:t=pos(0)+1:gosub getcmd if i\$>" " then open"I", #1, i\$+".crv" input#1,d,1,m:redim i\$(m-1),g(m,1):a=360/d for i=0 to m-1: input#1, i\$(i): for j=0 to 1: input#1, g(i,j): next j, i input#1, w:redim w(w):for i=1 to w:input#1, w(i):next i input#1,s,n,x,y:close end if if a\$="]" then goto crysave if a\$=chr\$(0) then goto program locate csrlin, pos(0),1 drctns: print"number of directions"; t=pos(0)+1:print d; :gosub getcmd if i\$>" " then d=val(i\$):a=360/d if a\$=chr\$(0) then goto program if a\$="]" then goto begin dvsn: print"max cell division";

'This program is QDRAGON.BAS, a QuickBASIC program for drawing fractals.

Inputs to this program are the files with a .CRV suffix. Rules for running it are in QRULES.TXT.

```
cellno: print"number of cells";
t=pos(0)+1:print m;:gosub getcmd
if i$>" " then m=val(i$):redim g(m-1,1),i$(m-1)
if a$=chr$(0) then goto program
if a$="]" then goto dvsn
i=0:j=1
gnetic:if i$(i)=" " then i$(i)="TA"+str$((i mod d)*a)+"R"
if g(i,0)=0 then g(i,0)=1
drawcode: print"draw code for cell"i"is";
t=pos(0)+1:print" "i\$(i)" <*> = 'do nothing'; <-> = 'invisible' ";
gosub getcmd
if i$>" " then
if instr(i$,"*") then i$(i)="TA"+str$((i mod d)*a)+"NBR":goto drawcode1
if instr(i$,"-") then i$(i)="TA"+str$((i mod d)*a)+"BR":goto drawcode1
i$(i)=i$
end if
drawcode1: if a$=chr$(0) then goto program
if a$="]" then
i=i-1:if i<0 then goto cellno else j=g(i,0):goto loop
end if
j=1
loop: print" cell"i"'s number"j"child is";
t=pos(0)+1:print g(i,j);
if j=g(i,0) then
if (i mod d)=0 then print" (end this entry with (*) to abort skip)";
end if
gosub getcmd
if i$>" " then
g(i,j)=val(i):if instr(i,".") then g(i,0)=j
end if
if a$=chr$(0) then goto program
if a$="]" then
if j=1 then goto gnetic
j=j-1:goto loop
end if
j=j+1:if j <=g(i,0) then goto loop
if (i mod d) <> 0 then i=i+1:goto ilupend
if i+d<=m then
if i$=" " then i=i+1:goto ilupend
if instr(i$,"*") then i=i+1:goto ilupend
for u=i+1 to i+d-1
if i$(i)=" " then i$(u)=" ":goto 10
if instr(i\$(i),"B") then i\$(u)="TA"+str\$((u mod d)*a)+"BR":goto 10
i\$(u)="TA"+str\$((u mod d)*a)+"R"
10 g(u,0)=g(i,0): for v=1 to g(i,0)
g(u,v)=g(i,v)-(g(i,v) \mod d)+((g(i,v)+u) \mod d)
next v,u:i=i+d:goto ilupend
end if
i=i+1
ilupend: if i<m then j=1:goto gnetic
initword:print"birth cell list length";
t=pos(0)+1:print w; :gosub getcmd
if i$>" " then w=val(i$):redim w(w):goto bcells
if a$=chr$(0) then goto program
if a$="]" then i=m-1:j=g(i,0):goto loop
bcells: i=1:print"cells"
loop1: print i"-th cell:";
t=pos(0)+1:print w(i);:gosub getcmd
if i$>" " then w(i)=val(i$)
if a$=chr$(0) then goto program
if a$="]" then if i=1 then goto initword else i=i-1:goto loop1
i=i+1:if i<=w then goto loop1
order:print"number of day's growth";:t=pos(0)+1:print n;:gosub getcmd
if i$>" " then n=val(i$)
if a$=chr$(0) then goto program
if a$="]" then i=w:goto bcells
```

```
length: print" cell length";
t=pos(0)+1:prints;:gosub getcmd
if i$>" " then s=val(i$)
if a$=chr$(0) then goto program
if a$="]" then goto order
xpos:print"x-origin";
t=pos(0)+1:print x;:gosub getcmd
if i$>" " then x=val(i$)
if a$=chr$(0) then goto program
if a$="]" then goto length
ypos:print"y-origin";
t=pos(0)+1:print y; :gosub getcmd
if i$>" " then y=val(i$)
if a$="]" then goto xpos
if a$=chr$(0) then goto program
crvsave: print"save name? (<cr> or <]> if no) ";:t=pos(0)+1:gosub getcmd
if i$="]" then goto ypos
if i$>" " then
open"0",1,i$+".crv"
print#1,d;1;m
for i=0 to m-1:print#1, i$(i):for j=0 to 1:print#1,g(i,j);
next j:print#1,:next i
print#1,w;:for i=1 to w:print#1,w(i);:next i:print#1,
print#1,s;n;x;y:close
end if
if a$="]" then goto ypos
if a$=chr$(0) then goto program
goto begin
program:screen 2:cls:dim k(n),cell(n)
draw"BM="+varptr$(x)+",="+varptr$(y)
for i=2 to w+1:cell(n)=w(i-1):draw"C="+varptr$(i):gosub dragon:next i
erase k.cell:goto crysave
dragon:
if n=0 then draw i$(cell(n))+str$(s):return
k(n)=1: while k(n) <= g(cell(n), 0): cell(n-1)= g(cell(n), k(n)): n=n-1
gosub dragon:n=n+1:k(n)=k(n)+1:wend
return
getcmd:i$=" ":t0=pos(0)-1:locate csrlin,t,1
getcmd1: a$=" ":a$=input$(1)
if instr(chr$(13)+"]"+chr$(0),a$) then print:return
if a$=chr$(8) then
if pos(0)=t then goto getcmd1
i$=left$(i$,len(i$)-1):locate csrlin,pos(0)-1,1:goto getcmd1
end if
I$=I$+a$:print a$;
if len(i$)=1 then print string$(t0-t+2,32);:locate csrlin,t+1,1
goto getcmd1
```

INDEX.PAS Contributed by Dick Pountain. Accompanies "Focus on Algorithms: Search and Destroy," August 1987, page 257.

```
{ INDEX.PAS in Turbo Pascal 3.0 for IBM PC and compatibles }
{ A book indexing program -- requires an input file -- execute as .COM }
{ Also requires a Boring Words dictionary, BORING.DIC, in the .COM file's directory }
{ To execute .COM file, enter "index <inputfilename> <outputfilename> }
program INDEX;
const CR = #13;
const maxDict = 3750;
{ maximum allowable dictionary entries}
type letters = 'a'..'z';
wordtype= string[16];
nodeptr = nodetype;
nodetype= record
info: wordtype;
```

```
next: nodeptr
               end:
var inputFile, outputFile: text;
   inputFilename, outputFilename: string[127];
   chr, firstletter: char;
                                                                                                                { the array of 26 lists }
   sortList: array[letters] of nodeptr;
   i: letters;
   word: wordtype;
   boringWords: array [1..maxDict] of wordtype;
   dictionary: text;
     endDict: integer;
procedure InitFiles;
                            { open input and output files }
begin
  inputFilename := paramSTR(1);
  Assign(inputFile,inputFilename);
  Reset(inputFile);
  outputFilename := paramSTR(2);
  Assign(outputFile, outputFilename);
  Rewrite(outputFile);
end:
procedure GetWord(VAR infile: text; VAR word: wordtype);
                             { read a cleaned-up word from the input file }
begin
                                                                                                                 { initialize to blank }
  word := '';
  repeat
    read(infile,chr);
                                                                                                            { convert all to lowercase }
    if chr in ['A' .. 'Z']
    then chr := char(ord(chr)+32);
                                                                                                        only accept alpha characters
    if chr in ['a' .. 'z']
                                                                                                              { add to word being built }
    then word := word+chr;
  until (chr = ' ') or (chr = CR) or eof(infile)
procedure Place(VAR list: nodeptr; word: wordtype);
var p,q,newnode: nodeptr;
   found: boolean;
                                                                          { insert new word into list in sorted position only if unique }
begin
  q := nil;
                                                                                                             { p points to head of list }
  p := list;
  found := false;
                                                                                                                  { not end of list and ]
  while (p <> nil)
                                                                                                            { word not already here and
    and (not found)
                                                                                             { word alphabetically later than current
    and (word >= p .info) do
                                                                                                    { does this node contain our word?
     if p . info = word
                                                                                                            { yes! word is already here }
     then found := true
     else begin
                                                                                                                remember this node and
      q := p;
p := p^.next
                                                                                                                move on to the next one }
     end; {while}
                                                                                                              { word isn't already here }
  if not found
  then begin
                                                                                                                    { create a new node
     New(newnode);
                                                                                                           { put word in its info field
    newnode . info := word;
                                                                                                                       { list was empty }
     if q = nil
     then begin
                                                                                                               { newnode becomes first }
       newnode .next := list;
      list := newnode
     end
     else begin
                                                                                                                  { insert after node q }
       newnode .next := q .next;
       q .next := newnode
     end
  end
procedure SquirtOut(list: nodeptr; VAR outfile: text);
                             { send sorted list to output file }
begin
  while list <> nil
  begin
    writeln(outfile, list . info);
     list := list .next
  end
end;
procedure ReadDictionary;
var i: integer;
begin
  Assign(dictionary, 'BORING.DIC');
  Reset(dictionary);
  i := 1;
  repeat
```

```
readln(dictionary, boringWords[i]);
    i := i + 1
    until eof(dictionary) or (i > maxDict);
                                                                                                 {number of actual dictionary entries}
  endDict := i;
Close(dictionary)
end:
function Boring (word: wordtype): boolean;
var left, right, try, svleft, svright: integer;
begin
  left := 1;
  right := endDict;
  repeat
    svleft := left; svright := right;
    try := (left + right) div 2;
    if word < boringWords[try]
      then right := try - 1
      else left := try + 1;
    until (word = boringWords[try]) or (svleft > svright);
if word = boringWords[try]
    then Boring := true
   else Boring := false
end:
begin
                            { main program }
  InitFiles:
  ReadDictionary;
  for i := 'a' to 'z' do sortList[i] := nil;
                                                                                                           { initialize all the lists }
  while not eof(inputFile) do
  begin
    GetWord(inputFile,word);
    firstletter := word[1];
                                                                                                                   { get first letter }
    if not Boring (word)
                                                                                                            { put word in proper place }
      then Place(sortList[firstletter], word);
  end; {while}
  for i := 'a' to 'z' do SquirtOut(sortList[i],outputFile);
  writeln('Keywords are contained in ',outputFilename);
  Close(inputFile):
  Close(outputFile)
end.
```

OPS8085.ARI Contributed by Alex Lane. Accompanies the article "Simulating a Microprocessor," August 1987, page 161.

```
% Subject: OPS8085.ARI - from Alex Lane: "Simulating a Microprocessor"
comp_regs(Regname) :-
         retract(state(R,PC,SP,_)),
         reg(Regname, Place),
         arg(Place, R, Reg);
         Xis A-Reg,
         adjust_flags(A,X,_,Flags),
         asserta(state(R,PC,SP,Flags)).
acc_math_with_carry(Op, Regname) :-
         retract(state(R,PC,SP,flags(_,_,_,CY,_))), % get what we need
         arg(1,R,A),
                                                       % extract A from R
         reg(Regname, Place),
                                  % Place is location of Regname in R
         arg(Place, R, Reg),
                                  % extract register value
         T1 = .. [Op, Reg, CY],
                                  % set up additions/subtractions
         T2 = ... [Op, A, T1],
         X is T2,
                                  % evaluate
         adjust_flags(A,X,Y,Flags), %adjust for flags
         argrep(R,1,Y,NewR),
                                % replace register value in R
         asserta(state(NewR, PC, SP, Flags)).
acc_math(Op,Regname) :-
         retract(state(R,PC,SP,_)),
         arg(1,R,A),
         reg(Regname, Place), arg(Place, R, Reg),
         T1 = .. [Op, A, Reg],
         X is T1,
```

```
adjust_flags(A, X, Y, Flags),
         argrep(R,1,Y,NewR),
         asserta(state(NewR,PC,SP,Flags)).
reg_math(Op, Regname) :-
         retract(state(R,PC,SP,flags(_,_,CY,_))), arg(1,R,A),
         reg(Regname, Place), % Place is location of Regname in R arg(Place, R, Reg), % extract register value
         T1 = .. [Op, Reg, TemReg], % take advantage of Arity inc() and dec()
         call(T1),
         adjust_flags(A, TemReg, NewReg, flags1(Z,S,P,_,AC)),
         argrep(R,1,NewReg,NewR), % replace register value in R
         asserta(state(NewR,PC,SP,flags(Z,S,P,CY,AC))).
not_implemented. % some things are not worth doing.
undefined.
move(mem, mem).
move(mem,D):-
         retract(state(R,PS,SP,F)),
         arg(6,R,H),
         arg(7,R,L),
         get_mem(H,L,Data),
         argrep(R,D,Data,NewR),
         asserta(state(NewR,PC,SP,F)).
move(S,mem) :-
         state(R,PS,SP,F),
         arg(6,R,H),
         arg(7,R,L),
         arg(S,R,Data),
         put_mem(H,L,Data).
move(S,D):-
         retract(state(R,P,SP,F)),
          arg(S,R,S1),
          argrep(R,D,S1,NewR),
          asserta(state(NewR,P,SP,F)).
reg_ptr(6, mem).
reg_ptr(A,B):-Bis(A+2) mod 8.
op(0).
                             /* NOP */
op(1):-
                             /* LXI BC */
          retract(state(regs(A,_,,D,E,H,L),PC,SP,Flags)),
          get_mem(PC,C), Hi is PC + 1,
          get_mem(Hi,B),
          NewPC is PC + 2,
          asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
                              /* STAX B */
 op(2):-
          state(regs(A,B,C,_,_,_),_,_,),
          put_mem(B,C,A).
                               /* INX B */
 op(3) :-
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
          BC is 256 * B + C + 1,
          decompose(BC, NewB, NewC),
          asserta(state(regs(A,NewB,NewC,D,E,H,L),PC,SP,Flags)).
                                                                  /* INR B */
                               reg_math(inc,b),!.
 op(4):-
                                                                  /* DCR B */
                               reg_math(dec,b),!.
 op(5):-
 op(6):-
                               /* MVI B, data */
          retract(state(regs(A,_,C,D,E,H,L),PC,SP,Flags)),
          get_mem(PC,B),
          NewPC is PC + 1,
          asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
```

```
op(7):-
                              /* RLC */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(Z,S,P,_,AC))),
         CY is A mod 128,
         A1 is (2 * A + CY) mod 256,
         asserta(state(regs(A1,B,C,D,E,H,L),PC,SP,flags(Z,S,P,CY,AC))).
op(9):-
                              /* DAD B (CY only)*/
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(Z,S,P,_,AC))),
         NewLisL+C,
         Carry is NewL // 256,
         NewH is H + B + Carry,
         CY is NewH // 256,
         FL is NewL /\ 255,
         FH is NewH /\ 255,
         asserta(state(regs(A,B,C,D,E,FH,FL),PC,SP,flags(Z,S,P,CY,AC))).
op(10):-
                               /* LDAX B */
          retract(state(regs(_,B,C,D,E,H,L),PC,SP,Flags)),
          get_mem(B,C,A),
          asserta(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)).
                                /* DCX B */
op(11):-
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
          BC is 256 * B + C - 1,
          decompose(BC,B1,C1),
          asserta(state(regs(A,B1,C1,D,E,H,L),PC,SP,Flags)).
                                                                   /* INR C */
                               reg_math(inc,c),!.
op(12):-
                                                                   /* DCR C */
                               reg_math(dec,c),!.
op(13):-
                                /* MVI C, data */
op(14):-
          retract(state(regs(A,B,_,D,E,H,L),PC,SP,Flags)),
          get_mem(PC,C),
          NewPC is PC + 1,
          asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
                                /* RRC */
op(15):-
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(Z,S,P,_,AC))),
          CY is A mod 2,
          I1 is CY * 128,
          A1 is A / / 2 + I1,
          asserta(state(regs(A1,B,C,D,E,H,L),PC,SP,flags(Z,S,P,CY,AC))).
                                /* LXI DE */
 op(17):-
          retract(state(regs(A,B,C,_,_,H,L),PC,SP,Flags)),
          memory(PC,E),
          Hi is PC + 1,
          memory(Hi,D),
          NewPC is PC + 2,
          asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
                                /* STAX D */
 op(18):-
          state(regs(A,_,_,D,E,_,_),_,_,_),
          put_mem(D,E,A).
                                /* INX D */
 op(19):-
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
          DE is 256 * D + E + 1,
          D1 is DE // 256,
          E1 is DE mod 256,
          asserta(state(regs(A,B,C,D1,E1,H,L),PC,SP,Flags)).
                                                                   /* INR D */
                                reg_math(inc,d),!.
 op(20):-
                                                                    /* DCR D */
 op(21):-
                                reg_math(dec,d),!.
 op(22):-
                                /* MVI D, data */
          retract(state(regs(A,B,C,_,E,H,L),PC,SP,Flags)),
          memory(PC,D),
          NewPC is PC + 1,
           asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
```

```
op(23):-
                               /* RAL */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(Z,S,P,CY,AC))),
          A1 is (2 * A + CY) mod 256,
         NewCY is A mod 128,
         asserta(state(regs(A1,B,C,D,E,H,L),PC,SP,flags(Z,S,P,NewCY,AC))).
op(25):-
                               /* DAD D */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(Z,S,P,_,AC))),
         NewLisL+E,
         Carry is NewL // 256,
         NewH is H + D + Carry,
         CY is NewH // 256,
         FL is NewL /\ 255,
         FH is NewH /\ 255,
          asserta(state(regs(A,B,C,D,E,FH,FL),PC,SP,flags(Z,S,P,CY,AC))).
op(26):-
                               /* LDAX D */
         retract(state(regs(_,B,C,D,E,H,L),PC,SP,Flags)),
         get_mem(D,E,A),
         asserta(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)).
op(27):-
                               /* DCX D */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
         DE is 256 * D + E - 1,
         decompose(DE,D1,E1),
         asserta(state(regs(A,B,C,D1,E1,H,L),PC,SP,Flags)).
op(28):-
                                                                  /* INR E */
                               reg_math(inc,e),!.
op(29):-
                               reg_math(dec,e),!.
                                                                  /* DCR E */
                               /* MVI E, data */
op(30):-
         retract(state(regs(A,B,C,D,_,H,L),PC,SP,Flags)),
         get_mem(PC,E),
         NewPC is PC + 1,
         asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
op(31):-
                               /* RAR */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(Z,S,P,CY,AC))),
         A1 is (A//2) + (128 * CY),
         NewCY is A mod 2,
         asserta(state(regs(A1,B,C,D,E,H,L),PC,SP,flags(Z,S,P,NewCY,AC))).
op(32):-
                               /* RIM (read interrupt mask) */
         not_implemented.
op(33):-
                               /* LXI HL */
         retract(state(regs(A,B,C,D,E,_,_),PC,SP,Flags)),
         get_mem(PC,L),
         Hi is PC + 1,
         get_mem(Hi,H),
         NewPC is PC + 2,
         asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
op(34):-
                               /* SHLD (store H L direct) */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
         get_mem(PC,Lo),
                                         NextPC is PC + 1,
         get_mem(NextPC,Hi),
         put_mem(Hi,Lo,L),
                                         NextLo is Lo + 1,
         put_mem(Hi, NextLo, H),
         NewPC is PC + 2.
         asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
op(35):-
                               /* INX H */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
         HL is 256 * H + L + 1,
         decompose(HL, H1, L1),
         asserta(state(regs(A,B,C,D,E,H1,L1),PC,SP,Flags)).
op(36):-
                               reg_math(inc,h),!.
                                                                  /* INR H */
op(37):-
                               reg_math(dec,h),!.
                                                                  /* DCR H */
```

```
op(38):-
                               /* MVI H, data */
         retract(state(regs(A,B,C,D,E,_,L),PC,SP,Flags)),
         get_mem(PC,H),
         NewPC is PC + 1,
         asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
                               /* DAA */
op(39):-
         state(regs(A,B,C,D,E,H,L),_,,flags(Z,S,P,CY,AC)),
         LSB is A /\ 15,
         LSB > 9;
         AC is 1, !, retract(state(_,PC,SP,F)),
         NewA is A + 6,
         asserta(state(regs(NewA,B,C,D,E,H,L),PC,SP,F)),
         MSB is NewA /\ 240,
         MSB > 9;
         CY is 1, !, retract(_,PC,_,_),
         FinalA is NewA + 6,
         asserta(state(regs(FinalA,B,C,D,E,H,L),PC,SP,F)).
op(41):-
                               /* DAD H */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(Z,S,P,_,AC))),
         NewL is L + L,
                                     % sum L plus lo-order register
         Carry is NewL // 256,
                                     % Carry is one if NewL > 255
         NewH is H + H + Carry,
                                     % sum H with hi-order register
         CY is NewH // 256,
                                     % CY flag is one if NewH > 255
         FL is NewL /\ 255,
                                      % bring into byte range
         FH is NewH /\ 255,
                                     % this one too
         asserta(state(regs(A,B,C,D,E,FH,FL),PC,SP,flags(Z,S,P,CY,AC))).
op(42):-
                               /* LHLD (load H L direct) */
         retract(state(regs(A,B,C,D,E,_,_),PC,SP,Flags)),
         get_mem(PC,Lo),
                                        NextPC is PC + 1,
         get_mem(NextPC,Hi),
         get_mem(Hi,Lo,L),
                                        NextLo is Lo + 1,
         get_mem(Hi, NextLo, H),
         NewPC is PC + 2,
         asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
op(43):-
                               /* DCX H */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
         HL is 256 * H + L - 1,
         decompose(HL, H1, L1),
         asserta(state(regs(A,B,C,D,E,H1,L1),PC,SP,Flags)).
                                                                   /* INR L */
op(44):-
                               reg_math(inc,1),!.
op(45):-
                                                                   /* DCR L */
                               reg_math(dec,1),!.
op(46):-
                               /* MVI L, data */
         retract(state(regs(A,B,C,D,E,H,_),PC,SP,Flags)),
         get_mem(PC,L),
         NewPC is PC + 1,
         asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
op(47):-
                               /* CMA (complement accumulator) */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
         bit_xor(A,255, NewA),
         asserta(state(regs(NewA,B,C,D,E,H,L),PC,SP,Flags)).
op(48):-
                               /* SIM (set interrupt mask) */
         not_implemented.
op(49):-
                               /* LXI SP */
         retract(state(Regs, PC,_,Flags)),
         get_mem(PC,SPL),
         Hi is PC + 1,
         get_mem(Hi,SPH),
         SP is 256 * SPH + SPL,
         NewPC is PC + 2,
         asserta(state(Regs, NewPC, SP, Flags)).
```

```
op(50):-
                                /* STA Adr (store accumulator in address) */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
          get_mem(PC,Lo),
                                         HiAdr is PC + 1,
          get_mem(HiAdr, Hi),
          put_mem(Hi,Lo,A),
          NewPC is PC + 2,
          asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
op(51):-
                                /* INX SP */
          retract(state(Regs, PC, SP, Flags)),
          NewSP is SP + 1,
          check_overflow(NewSP, FinalSP),
          asserta(state(Regs, PC, FinalSP, Flags)).
op(52):-
                               /* INR M */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(_,_,_,CY,_))),
          get_mem(H,L,Data),
          NewData is Data + 1,
          adjust_flags(A,NewData,FinalData,flags1(Z,S,P,_,AC)),
          put_mem(H,L,FinalData),
          asserta(state(regs(A,B,C,D,E,H,L),PC,SP,flags(Z,S,P,CY,AC))).
op(53):-
                               /* DCR M */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(_,_,_,CY,_))),
          get_mem(H,L,Data),
          NewData is Data - 1,
          adjust_flags(A,NewData,FinalData,flags1(Z,S,P,_,AC)),
          put_mem(H,L,FinalData),
          asserta(state(regs(A,B,C,D,E,H,L),PC,SP,flags(Z,S,P,CY,AC))).
op(54):-
                               /* MVI M, data */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
          get_mem(PC, Data),
         put_mem(H, L, Data),
         NewPC is PC + 1,
         asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
op(55):-
                               /* STC (set carry) */
         retract(state(Regs, PC, SP, flags(Z, S, P, _, AC))),
         asserta(state(Regs, PC, SP, flags(Z,S,P,1,AC))).
op(57):-
                               /* DAD SP */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(Z,S,P,_,AC))),
         decompose(SP,SPH,SPL),
         NewL is L + SPL,
         Carry is NewL // 256,
         NewH is H + SPH + Carry,
         CY is NewH // 256,
         FL is NewL /\ 255,
         FH is NewH /\ 255,
         asserta(state(regs(A,B,C,D,E,FH,FL),PC,SP,flags(Z,S,P,CY,AC))).
op(58):-
                               /* LDA Adr */
         retract(state(regs(_,B,C,D,E,H,L),PC,SP,Flags)),
         get_mem(PC,Lo),
         NextPC is PC + 1,
         get_mem(NextPC,Hi),
         get_mem(Hi,Lo,A),
         NewPC is PC + 2,
         asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
op(59):-
                               /* DCX SP */
         retract(state(Registers, PC, SP, Flags)),
         NewSP is SP - 1,
         check_overflow(NewSP, FinalSP),
         asserta(state(Registers, PC, FinalSP, Flags)).
op(60):-
                               reg_math(inc,a),!.
                                                                  /* INR A */
op(61):-
                               reg_math(dec,a),!.
                                                                  /* DCR A */
op(62):-
                               /* MVI A, Data */
         retract(state(regs(_,B,C,D,E,H,L),PC,SP,Flags)),
         get_mem(PC,A),
         NewPC is PC + 1,
         asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
```

```
op(63):-
                               /* CMC (complement carry) */
         retract(state(Regs, PC, SP, flags(Z, S, P, CY, AC))),
         bit_xor(1,CY,NewCY),
         asserta(state(Regs,PC,SP,flags(Z,S,P,NewCY,AC))).
                                 /* MOV Destination, Source */
op(Code) :-
         Code > 63, Code < 128,
         B210 is Code /\ 7,
                                          % decode which reg is in bits 0-2
                                          % do same for bits 3-5
         B543 is (Code /\ 56) >> 3,
         reg_ptr(B210,S),
         reg_ptr(B543,D),
         move(S,D).
op(128):-
                      acc_math(+,b),!.
op(129):-
                      acc_math(+,c),!.
op(130):-
                      acc_math(+,d),!.
                      acc_math(+,e),!.
op(131):-
                      acc_math(+,h),!.
op(132):-
                      acc_math(+,1),!.
op(133):-
op(134):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         get_mem(H,L,Q),
         X is A + Q,
         adjust_flags(A,X,Y,Flags),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,Flags)).
                      acc_math(+,a),!.
op(135):-
                                                                   % add B with carry
                      acc_math_with_carry(+,b),!.
op(136):-
op(137):-
                      acc_math_with_carry(+,c),!.
                      acc_math_with_carry(+,d),!.
op(138):-
                       acc_math_with_carry(+,e),!.
op(139) :-
op(140):-
                      acc_math_with_carry(+,h),!.
op(141):-
                       acc_math_with_carry(+,1),!.
op(142):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         get_mem(H,L,Q),
         X is A + Q,
         adjust_flags(A, X, Y, Flags),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,Flags)).
                       acc_math_with_carry(+,a),!.
op(143) :-
op(144):-
                       acc_{math(-,b),!}
op(145):-
                       acc_math(-,c),!.
op(146):-
                       acc_math(-,d),!.
op(147):-
                       acc_math(-,e),!.
op(148):-
                       acc_math(-,h),!.
op(149) :-
                       acc_math(-,1),!.
op(150):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
          get_mem(H,L,Q),
         X is A + Q,
         adjust_flags(A, X, Y, Flags),
          asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,Flags)).
op(151):-
                       acc_math(-,a),!:
```

```
op(152):-
                      acc_math_with_carry(-,b),!.
                      acc_math_with_carry(-,c),!.
op(153) :-
                      acc_math_with_carry(-,d),!.
op(154):-
                      acc_math_with_carry(-,e),!.
op(155):-
op(156):-
                      acc_math_with_carry(-,h),!.
op(157):-
                      acc_math_with_carry(-,1),!.
op(158):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(_,_,_,CY,_))),
         get_mem(H,L,Q),
         X is A - Q - CY,
         adjust_flags(A, X, Y, Flags),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,Flags)).
                      acc_math_with_carry(-,a),!.
op(159) :-
op(160):-
                      acc_{math(/\,b),!}
                      acc_{math(//,c),!}
op(161):-
op(162):-
                      acc_math(/\,d),!.
op(163):-
                      acc_math(/\,e),!.
op(164) :-
                      acc_math(/\,h),!.
op(165):-
                      acc_{math(/\,1),!}
op(166):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         get_mem(H,L,M),
         Xis A /\ M,
         adjust_flags(A,X,Y,flags(Z,S,P,_,_)),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,1))).
                               % AND A (affects flags)
op(167) :-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         adjust_flags(A,A,Y,flags(Z,S,P,_,_)),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,1))).
op(168):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         bit_xor(A,B,X),
         adjust_flags(A,X,Y,flags(Z,S,P,_,_)),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,0))).
op(169):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         bit_xor(A,C,X),
         adjust_flags(A,X,Y,flags(Z,S,P,_,_)),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,0))).
op(170):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         bit_xor(A,D,X),
         adjust_flags(A,X,Y,flags(Z,S,P,_,_)),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,0))).
op(171):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         bit_xor(A,E,X),
          adjust_flags(A,X,Y,flags(Z,S,P,_,_)),
          asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,0))).
op(172):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
          bit_xor(A,H,X),
          adjust_flags(A, X, Y, flags(Z, S, P, _, _)),
          asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,0))).
```

```
op(173):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         bit_xor(A,L,X),
         adjust_flags(A,X,Y,flags(Z,S,P,_,_)),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,0))).
op(174):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         get_mem(H,L,M),
         bit_xor(A,M,X),
         adjust_flags(A,X,Y,flags(Z,S,P,_,_)),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,0))).
op(175):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         bit_xor(A,A,X),
         adjust_flags(A, X, Y, flags(Z, S, P, _, _)),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,0))).
                      acc_math(/\,b),!.
op(176):-
op(177):-
                      acc_math(/\,c),!.
op(178):-
                      acc_math(/\,d),!.
op(179) :-
                               acc_math(/\,e),!.
                                                                 /* ORA E */\
op(180):-
                               acc_math(/\,h),!.
                                                                 /* ORA H */
op(181):-
                               acc_math(/\,1),!.
op(182):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         get_mem(H,L,M),
         XisA/\M,
         adjust_flags(A, X, Y, flags(Z, S, P,_,_)),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,0))).
op(183):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         adjust_flags(A,A,Y,flags(Z,S,P,_,_)),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,0))).
                      comp_regs(b),!.
                                                      /* CMP B */
op(184):-
op(185):-
                                                      /* CMP C */
                      comp_regs(c),!.
                                                      /* CMP D */
op(186):-
                      comp_regs(d),!.
                      comp_regs(e),!.
                                                      /* CMP E */
op(187):-
op(188):-
                      comp_regs(h),!.
                                                      /* CMP H */
op(189):-
                      comp_regs(1),!.
                                                      /* CMP L */
op(190):-
                                /* CMP M */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         get_mem(H,L,Q),
         X is A - Q,
         adjust_flags(A,X,_,Flags),
         asserta(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)).
op(191):-
                               /* CMP A */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         adjust_flags(A,O,_,Flags),
         asserta(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)).
                                /* RNZ */
op(192):-
         (not zero_flag_is_set, return); true.
                               /* POP B */
op(193):-
         retract(state(regs(A,_,_,D,E,H,L),PC,SP,Flags)),
         get_mem(SP,C),
         Hi is SP + 1,
         get_mem(Hi,B),
```

```
NewSP is SP + 2.
          asserta(state(regs(A,B,C,D,E,H,L),PC,NewSP,Flags)).
op(194) :-
                                 /* JNZ */
          (not zero_flag_is_set, jump); carry_on.
op(195):-
                                 /* JMP */
          jump.
op(196):-
                                 /* CNZ */
          (not zero_flag_is_set, call); carry_on.
op(197) :-
                                 /* PUSH B */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
          Hi is SP - 1,
          put_mem(Hi,B),
          Lo is SP - 2,
          put_mem(Lo,C),
          NewSP is SP - 2,
          asserta(state(regs(A,B,C,D,E,H,L),PC,NewSP,Flags)).
op(198) :-
                                 /* ADI D8 */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
          get_mem(PC,Q),
          X is A + Q,
          adjust_flags(A,X,Y,Flags),
          NewPC is PC + 1,
          asserta(state(regs(Y,B,C,D,E,H,L),NewPC,SP,Flags)).
op(199):-
                                 /* RST 0 */
         reset(0).
op(200):-
                                 /* RZ */
         (zero_flag_is_set, return); true.
                                 /* RET */
op(201):-
         return.
op(202):-
                                 /* JZ */
          (zero_flag_is_set, jump); carry_on.
op(204):-
                                 /* CZ */
         (zero_flag_is_set, call); carry_on.
op(205):-
                                 /* CALL */
         call.
op(206):-
                                 /* ACI D8 */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(_,_,_,CY,_))),
         get_mem(PC,Q),
         X is A + Q + CY,
         adjust_flags(A,X,Y,Flags),
         NewPC is PC + 1,
         asserta(state(regs(Y,B,C,D,E,H,L),NewPC,SP,Flags)).
op(207):-
                                 /* RST 1 */
         reset(1).
op(208):-
                                /* RNC */
         (not carry_flag_is_set, return); true.
op(209):-
                                 /* POP D */
         retract(state(regs(A,B,C_,_,H,L),PC,SP,Flags)),
         get_mem(SP,E),
         Hi is SP + 1,
         get_mem(Hi,D),
         NewSP is SP + 2,
         asserta(state(regs(A,B,C,D,E,H,L),PC,NewSP,Flags)).
op(210):-
                                /* JNC */
         (not carry_flag_is_set, jump); carry_on.
op(211):-
                                 /* OUT */
         not_implemented.
```

```
/* CNC */
op(212):-
         (not carry_flag_is_set, call); carry_on.
                                 /* PUSH D */
op(213):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
         Hi is SP - 1,
         put_mem(Hi,D),
         Lo is SP - 2,
         put_mem(Lo,E),
         NewSP is SP - 2,
         asserta(state(regs(A,B,C,D,E,H,L),PC,NewSP,Flags)).
                                 /* SUI D8 */
op(214) :-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         get_mem(PC,Q),
         X is A - Q,
         adjust_flags(A,X,Y,Flags),
         NewPC is PC + 1,
         asserta(state(regs(Y,B,C,D,E,H,L),NewPC,SP,Flags)).
op(215):-
                                 /* RST 2 */
         reset(2).
                                 /* RC */
op(216):-
         (carry_flag_is_set, return); true.
                                 /* JC */
op(218):-
          (carry_flag_is_set, jump); carry_on.
                                 /* IN */
op(219):-
         not_implemented.
                                 /* CC */
op(220):-
          (carry_flag_is_set, call); carry_on.
op(222):-
                                 /* SBI */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(_,_,_,CY,_))),
          get_mem(PC,Q),
          X is A - Q - CY,
          adjust_flags(A, X, Y, Flags),
          NewPC is PC + 1,
          asserta(state(regs(Y,B,C,D,E,H,L),NewPC,SP,Flags)).
                                 /* RST 3 */
op(223):-
          reset(3).
                                 /* RPO odd parity; flag is 0 */
op(224):-
          (not parity_flag_is_set, return); true.
                                 /* POP H */
op(225):-
          retract(state(regs(A,B,C,D,E,_,_),PC,SP,Flags)),
          get_mem(SP,L),
          Hi is SP + 1,
          get_mem(Hi,H),
          NewSP is SP + 2,
          asserta(state(regs(A,B,C,D,E,H,L),PC,NewSP,Flags)).
                                 /* JPO odd parity; flag is 0 */
op(226):-
          (not parity_flag_is_set, jump); carry_on.
op(227):-
                                 /* XTHL */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
          get_mem(SP, NewH),
          put_mem(SP,H),
          SP1 is SP + 1,
          get_mem(SP1, NewL),
          put_mem(SP1,L),
          asserta(state(regs(A,B,C,D,E,NewH,NewL),PC,SP,Flags)).
                                 /* CPO odd parity; flag is 0 */
op(228):-
          (not parity_flag_is_set, call); carry_on.
op(229):-
                                 /* PUSH H */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
          Hi is SP - 1,
```

```
put_mem(Hi,H),
          Lo is SP - 2,
          put_mem(Lo,L),
          NewSP is SP - 2,
          asserta(state(regs(A,B,C,D,E,H,L),PC,NewSP,Flags)).
 op(230):-
                                  /* ANI D8 */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
          get_mem(PC,M),
          XisA/\M,
          adjust_flags(A, X, Y, Flags),
          asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,Flags)).
 op(231):-
                                 /* RST 4 */
          reset(4).
                                 /* RPE odd parity; flag is 1 */
op(232):-
          (parity_flag_is_set, return); true.
op(233):-
                                  /* PCHL */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
          decompose(PC,PCH,PCL), %(i,o,o)
decompose(NewPC,H,L), %(o,i,i)
          asserta(state(regs(A,B,C,D,E,PCH,PCL),NewPC,SP,Flags)).
op(234):-
                                 /* JPE odd parity; flag is 1 */
          (parity_flag_is_set, jump); carry_on.
op(235):-
                                 /* XCHG */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)),
          asserta(state(regs(A,B,C,H,L,D,E),PC,SP,Flags)).
                                 /* CPE odd parity; flag is 1 */
op(236):-
          (parity_flag_is_set, call); carry_on.
op(238):-
                                 /* XRI D8 */
          retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
          get_mem(PC,M),
          bit_xor(A,M,X),
          adjust_flags(A,X,Y,flags(Z,S,P,_,_)),
          asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,0))).
op(239):-
                                 /* RST 5 */
          reset(5).
op(240):-
                                 /* RP sign flag is 0 */
          (not sign_flag_is_set, return); true.
op(241) :-
                                 /* POP PSW */
          retract(state(regs(_,B,C,D,E,H,L),PC,SP,_)),
          get_mem(SP,Flags),
          Hi is SP + 1,
          get_mem(Hi,A),
          NewSP is SP + 2,
         Sis (Flags /\ 128) / 128,
         Z is (Flags /\ 64) / 64,
          AC is (Flags /\ 16) / 16,
         P is (Flags / \ 4) / 4,
         CY is Flags /\ 1,
         asserta(state(regs(A,B,C,D,E,H,L),PC,NewSP,flags(Z,S,P,CY,AC))).
op(242):-
                                 <* JP sign flag is 0 */
         (not sign_flag_is_set, jump); carry_on.
op(243):-
                                 /* DI */
         not_implemented.
op(244):-
                                 /* CP sign flag is 0 */
         (not sign_flag_is_set, call); carry_on.
op(245):-
                                 /* PUSH PSW */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,flags(Z,S,P,CY,AC))),
         Hi is SP - 1,
         put_mem(Hi,A),
         Lo is SP - 2,
```

```
Flags is CY * 1 + P * 4 + AC * 16 + Z * 64 + S * 128,
         put_mem(Lo, Flags),
         NewSP is SP - 2.
         asserta(state(regs(A,B,C,D,E,H,L),PC,NewSP,flags(Z,S,P,CY,AC))).
                                /* ORI D8 */
op(246):-
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         get_mem(PC,M),
         XisA/\M,
         adjust_flags(A, X, Y, flags(Z,S,P,_,_)),
         asserta(state(regs(Y,B,C,D,E,H,L),PC,SP,flags(Z,S,P,0,0))).
op(247):-
                                 /* RST 6 */
         reset(6).
                                /* RM sign flag is 1 AR/
op(248):-
         (sign_flag_is_set, return); true.
op(249):-
                                /* SPHL */
         retract(state(regs(A,B,C,D,E,H,L),PC,_,Flags)),
         decompose(SP,H,L),
         asserta(state(regs(A,B,C,D,E,H,L),PC,SP,Flags)).
                                /* JM sign flag is 1 */
op(250):-
         (sign_flag_is_set, jump); carry_on.
                                 /* EI */
op(251):-
         not_implemented.
                                 /* CM sign flag is 1 */
op(252):-
         (sign_flag_is_set, call); carry_on.
op(254):-
                                 /* CPI D8 */
         retract(state(regs(A,B,C,D,E,H,L),PC,SP,_)),
         get_mem(PC,Q),
         X is A - Q,
         adjust_flags(A, X, Y, Flags),
         NewPC is PC + 1,
         asserta(state(regs(A,B,C,D,E,H,L),NewPC,SP,Flags)).
                                /* RST 7 */
op(255):-
         reset(7).
op(_) :- write('undefined opcode'), nl.
% end: OPS8085.ARI
```

HELP80.ARI Contributed by Alex Lane. Accompanies the article "Simulating a Microprocessor," August 1987, page 161.

```
% File: HELP80.ARI
   "nondestructive" help - saves the screen while you access help info.
%
   There's nothing here that can't be done with most Prolog implementations; here, we use Arity's region_ca/\3,
%
   tget/2, and tmove/3 predicates to save the screen, save the cursor position, and restore the cursor position.
%
   If your Prolog doesn't implement these features, they can be excised with no great harm done.
   Arity/Prolog encloses strings within '$' characters, so this part may require some minor rewrite to work with other
%
%
   Prolog implementations.
%
help :- grab(S,R,C), write($HELP
Commands can be entered as one or two unique letters ('t' for 'trace', 'sh'
for 'show'), or as complete words ('step').
All numbers input by the user are assumed to be base 16 (hexadecimal) numbers. This means, for example, that the
command 'step 10' is a request for 16 steps, not 10!
```

Hexadecimal numbers that start with a letter must be typed in with a leading zero (e.g., 'ff' must be typed in as 'Off').

Additional help is available for the following commands:

status change trace quit show reset step

To get additional help, type 'help' and the command word you are interested in.

\$), let_go(S,R,C).

help(status) :- grab(S,R,C), write(\$STATUS

This command prints the state/4 clause to the screen. The information displayed by state/4 is not as "pretty" as that shown by the 'show' command, but is complete.

\$), let_go(S,R,C).

help(help) :- help.

help(show) :- grab(S,R,C), write(\$SHOW

Use this command to display the contents of a register, of a register-pair pointer, and of memory.

To display the contents of a register or of a register pair, type 'show' followed by the appropriate name.

If no parameters are supplied, 16 bytes of memory are shown starting at the current PC value. If only one memory address is specified, only the contents of that address will be displayed.

Examples:

show b Show contents of register b

show bc Show contents of address pointed to by bc

show Show contents of PC through PC + 16 (decimal)

show 0020 Show contents of address 0020

show 0020 002f Show contents of addresses 0020 through 002f

\$), let_go(S,R,C).

help(change) :- grab(S,R,C), write(\$CHANGE

Use this command to change the contents of a register, of an address in memory, or of an address pointed to by a 16-bit register.

To change something, type 'change' followed by the appropriate register name or memory location, followed in turn by the new value.

Examples:

change 002c 3f Changes the contents of address 002c to 3f change d 2c Changes the contents of register d to 2c

change pc 0f3 Changes the contents of the program counter to 0f3

\$), let_go(S,R,C).

help(reset) :- grab(S,R,C), write(\$RESET

This command resets everything back to the initial state:

state(regs(0,0,0,0,0,0,0),0,255,flags(0,0,0,0,0))

\$), let_go(S,R,C).

help(step) :- grab(S,R,C), write(\$STEP

This command causes an instruction to be executed. Multiple instructions can be stepped by adding a second, numeric argument.

Examples:

stepSteps through one instructionstep 3Steps through three instructionsstep 10Steps through 16 instructions

\$), let_go(S,R,C).

help(my_trace) :- grab(S,R,C), write(\$TRACE

Like 'step', this command causes one instruction to be executed, but also displays the processor state upon completion. Multiple traces can be performed by adding a second, numeric argument.

```
Examples:
```

```
Steps through an instruction and shows all
          trace
          trace 3
                                    Steps and shows all through three instructions
                                    Steps and shows all through 16 instructions
          trace 10
$), let_go(S,R,C).
help(quit) :- grab(S,R,C), write($QUIT
This command brings you back to Prolog's '?-' prompt.
$), let_go(S,R,C).
%
   grab/1 takes advantage of an Arity/Prolog predicate that reads characters and their attributes from the screen and places
     them in a string. This in effect saves the screen while you are off getting help.
%
   let_go/1 waits for the user to strike a key and then restores the screen to the way it looked before the help session.
   if you don't mind losing the screen contents when you seek help, you can dispense with these predicates.
%
grab(S,R,C):-tget(R,C),
               region_ca((0,0),(24,79),S),
               cls. !.
let_go(S,R,C) :- write($Strike a key to continue ... $),
                 getO(Ch),
                 cls.
                 region_ca((0,0),(24,79),S),
                 tmove(R,C), !.
% end
```

PREDS80.ARI Contributed by Alex Lane. Accompanies the article "Simulating a Microprocessor," August 1987, page 161.

```
% Subject: PREDS80.ARI - from Alex Lane: "Simulating a Microprocessor"
% bit_xor(A,B,C) :: C is exclusive-or of A and B
bit_xor(0,0,0):-!.
bit_xor(0,1,1):-!.
bit_xor(1,0,1):-!.
bit_xor(1,1,0):-!.
bit_xor(A,B,X):-
        AAO is A // 2,
                                 AO is A mod 2,
                                 BO is B mod 2,
        BBO is B / / 2,
        bit_xor(AAO,BBO,XXO),
        bit_xor(A0,B0,X0),
        X is 2 * XX0 + X0.
% adjust_flags(In, Out, flags(Z,S,P,CY,AC)) :: In --> Out and flags checked.
adjust_flags(A,In,Out,flags(Z,S,P,CY,AC)):-
        check_carry(In, Out, CY),
        check_zero(Out,Z),
        check_parity(Out,P),
        check_aux_carry(A,Out,AC),
        check_sign(Out,S).
% carry_on :: PC <-- PC + 2.
```

```
carry on :-
         retract(state(Regs, PC, SP, Flags)),
         NewPC is PC + 2,
         asserta(state(Regs, NewPC, SP, Flags)).
% check_zero(A,B) :: A == 0 --> B == 1, else B == 0.
check_zero(0,1) :- !.
check_zero(_,0).
% check_parity(X,Y) :: Y reflects odd parity of number of bits in X.
check_parity(X,Y):-par(X,T), Y is T mod 2, !.
par(0,0):-!.
par(1,1):-!.
par(N,P) :- J is N mod 2,
            Kis N// 2,
            par(K,P1),
            PisJ+P1.
check\_sign(X,1) := X > 127, !.
check_sign(_,0).
check_carry(In,Out,1):- In > 255, Out is In - 255, !.
check_carry(In,Out,1) :- In < 0, Out is In + 255, !.
check_carry(In, In,0).
check_aux_carry(OldAccum, NewAccum, 1) :-
         OldAccum /\ 24 =:= 8,
                                      % old bit 3 on and old bit 4 off
         NewAccum / \ 24 = := 16,!.
                                      % new bit 4 on and new bit 3 off
                                 % (=:= evaluates both sides and tests for equality)
check_aux_carry(_,_,0) :-!.
zero_flag_is_set :-
          state(_,_,_,flags(1,_,_,_,_)),!.
sign_flag_is_set :-
          state(_,_,_,flags(_,1,_,_,_)),!.
parity_flag_is_set :-
 pstate(_,_,_,flags(_,_,1,_,_)),!.
carry_flag_is_set :-
          state(_,_,_,flags(_,_,_,1,_)),!.
aux_carry_flag_is_set :-
          state(_,_,_,flags(_,_,_,1)),!.
% reset (N) :: store PC on stack, PC <-- 8 * N.
%
reset(N) :-
     retract(state(Registers, PC, SP, Flags)),
     decompose(PC, PCH, PCL),
     Hi is SP - 1,
     put_mem(Hi, PCH),
     NewSP is SP - 2,
     put_mem(NewSP, PCL),
     NewPC is 8 * N,
     asserta(state(Registers, NewPC, NewSP, Flags)).
% return :: pop PC off stack, adjust SP.
return :-
        retract(state(Registers,_,SP,Flags)),
        get_mem(SP, PCL),
        Hi is SP + 1,
        get_mem(Hi, PCH),
        PC is 256 * PCH + PCL,
        NewSP is SP + 2,
        asserta(state(Registers, PC, NewSP, Flags)).
```

```
jump :-
       retract(state(Registers, PC, SP, Flags)),
       get_mem(PC, PCL),
       Hi is PC + 1,
       get_mem(Hi, PCH),
       NewPC is 256 * PCH + PCL,
       asserta(state(Registers, NewPC, SP, Flags)).
call :-
       retract(state(Registers, PC, SP, Flags)),
       PCN is PC + 2,
                                   % here? or on return?
       PCH is PCN // 256,
       PCL is PCN mod 256,
       SP1 is SP - 1,
       SP2 is SP - 2,
       put_mem(SP1, PCH),
       put_mem(SP2, PCL),
       get_mem(PC, NPCL),
       Hi is PC + 1,
       get_mem(Hi, NPCH),
       NewPC is 256 * NPCH + NPCL,
       asserta(state(Registers, NewPC, SP2, Flags)).
% put_mem(H, L, Data) :: store Data in Address <-- 256 * H + L.
put_mem(Hi, Lo, NewData):-
         Address is 256 * Hi + Lo,
         put_mem(Address, NewData).
% put_mem(H, L, Data) :: store Data in Address.
put_mem(Address, NewData) :-
        retract(memory(Address,_)),
        asserta(memory(Address, NewData)).
% get_mem( H, L, Data) :: fetch Data in Address <-- 256 * H + L.
get_mem(Hi, Lo, Data):-
        Address is 256 * Hi + Lo,
         get_mem(Address, Data).
% get_mem( Address, Data) :: fetch Data in Address.
get_mem(Address, Data) :-
        memory(Address, Data).
% decompose(Address, Hibyte, Lobyte) :: Address <-- 256 * Hibyte + Lobyte
    (0, 1, 1)
decompose(Address, Hibyte, Lobyte) :- %(o,i,i)
         var(Address),
                                      % if Address is uninstantiated
         Address is 256 * Hibyte + Lobyte.
   decompose(Address, Hibyte, Lobyte) :: Address --> Hibyte; Lobyte
%
      (1,0,0)
%
decompose(Address, Hibyte, Lobyte) :- %(i,o,o)
        Fis Address / 256,
        His integer(F),
        Hibyte is H /\ 255,
        Gis Address - 256 * H,
        Lobyte is integer(G).
                                   % write an address in hex
put_address(Lo) :-
     decompose(Lo, LoH, LoL),
     dec_hex_byte(LoH, LHH),
     dec_hex_byte(LoL, LLH),
     write(LHH), write(LLH), write(':'),!.
```

```
Decimal-to-hex and hex-to-decimal conversions.
dec_hex_byte(Dec, Hex):-
                                   % (i,o)
    var(Hex),!,
     Dec < 256,
     Hi is Dec >> 4,
     Lo is Dec /\ 15,
     d_h(Hi,H),
     d_h(Lo,L),
     list_text([H,L],Hex).
dec_hex_byte(Dec, Hex):-
                                   % (o,i)
     list_text([H,L],Hex),
     d_h(Hi,H),
     d_h(Lo,L),
    Dec is 16 * Hi + Lo,!.
d_h(In, Out):-
                         % (i,o)
    In < 10,
    Out is In + 48.
d_h(In, Out):-
                         % (i,o)
    Out is In +55.
d_h(Out, In):-
                         % (o,i)
    In < 65,
    Out is In - 48.
d_h(Out, In):-
                         % (o,i)
    In < 71,
    Out is In - 55.
min(I1,I1,I2):- I1 =< I2,!.
min(I2, I1, I2) :- I2 < I1.
for(X,X,X):-!.
for(X,Y,X).
for(X,Y,Z) := X1 is X + 1, for(X1,Y,Z).
append([H | T], L, [H | R]) :-!, append(T, L, R).
append([], L, L).
valid_adr(H,L):-top_of_memory(TOM), TOM>= (256 * H + L).
             reg(b,2).
                           reg(c,3).
reg(a,1).
                                         reg(d,4).
reg(e,5).
             reg(h,6).
                           reg(1,7).
f1(z,1).
             fl(s,2).
                           fl(p,3).
                                         fl(cy,4).
                                                       fl(ac,5).
% end: PREDS80.ARI
```

PARSE80.ARI Contributed by Alex Lane. Accompanies the article "Simulating a Microprocessor," August 1987, page 161.

```
% Subject: PARSE80.ARI - from Alex Lane: "Simulating a Microprocessor"
  Use DCG to parse a command (request) from the simple monitor.
%
  Intended to be used with TOKENS.ARI (5/10/86)
   Basically want to implement following commands:
%
                                                                                  Examples:
  change --+-- memory-address ---- value
                                                                                 (change 002c 3f)
                                                                                 (change d 2c)
           +-- register-contents ---- value
            +-- register-pair-pointer --- value
                                                                                 (change pc 0f3)
                                                                                 (help)
% help (a screen of help)
   quit (back to DOS)
                                                                                 (quit)
                                                                                 (reset)
   reset (all parameters to some intial state)
```

```
(show b)
%
   show --+-- register-contents
%
          +-- contents of register-pair-pointer
                                                                                  (show bc)
%
                                                                                  (show)
          +-- <memory, no range specified>
%
          +-- starting-memory-address
                                                                                  (show 0020)
%
          +-- starting-address --- ending-address
                                                                                  (show 0020 002f)
%
   step
                                                                                  (step)
%
                                                                                  (trace)
   trace
%
parse_tokens(change(Adr, Val)) --> verb(change), address(Adr), byte(Val).
parse_tokens(change(Reg, Val)) --> verb(change), register_name(Reg), byte(Val).
parse_tokens(change(RegPair, Val)) --> verb(change), pointer(RegPair), byte(Val).
parse_tokens(change(Flag, Value)) --> verb(change), flag(Flag), flag_value(Value).
parse_tokens(show(Reg)) --> verb(show), register_name(Reg).
parse_tokens(show(Group)) --> verb(show), group_designation(Group).
parse_tokens(show(RegPair)) --> verb(show), pointer(RegPair).
parse_tokens(show(Start,End)) --> verb(show), address(Start), address(End),
                                   { Start = < End }.
parse_tokens(show(Start)) --> verb(show), address(Start).
parse_tokens(help(Topic)) --> verb(help), verb(Topic).
parse_tokens(step(Times)) --> verb(step), byte(Times).
parse_tokens(my_trace(Times)) --> verb(my_trace), byte(Times).
parse_tokens(Command) --> verb(Command), { Command \== change }.
address(Adr) --> [Adr], { Adr>= 0, top_of_memory(TOM), Adr =< TOM }.
byte(Val) --> [Val], \{ Val >= 0, Val < 256 \}.
flag_value(X) \longrightarrow [X], \{X == 1; X == 0\}.
register_name(a) --> [a].
                             register_name(b) --> [b].
register_name(c) --> [c]. register_name(d) --> [d].
register_name(e) --> [e]. register_name(h) --> [h].
register_name(1) --> [1]. register_name(flags) --> [f]; [f1]; [flags].
flag(z) \longrightarrow [z]. flag(s) \longrightarrow [s]. flag(p) \longrightarrow [p].
flag(cy) \longrightarrow [cy]. flag(ac) \longrightarrow [ac].
group_designation(all) --> [al]; [all].
group_designation(regs) --> [r]; [re]; [regs].
group_designation(stack) --> [st];[stack].
pointer(bc) --> [bc].
                                pointer(de) --> [de].
pointer(pc) --> [p]; [pc].
                                pointer(sp) --> [s];[sp].
pointer(h1) --> [h1]; [m]; [me]; [mem].
verb(change) --> [c];[ch];[change];[set].
                                              % 'set' is a synonym
verb(status) --> [state].
                                              % must say 'state'
verb(help) --> [h]; [he]; [help].
verb(quit) --> [q]; [qu]; [quit].
verb(show) --> [sh]; [show].
verb(step) --> [st];[step].
verb(my_trace) --> [t];[tr];[trace].
verb(reset_processor) --> [r];[re];[reset].
% end PARSE80.ARI
```

MON80.ARI Contributed by Alex Lane. Accompanies the article "Simulating a Microprocessor," August 1987, page 161.

```
% Subject: MON80.ARI - from Alex Lane: "Simulating a Microprocessor"
% state(_) needs to have been asserted.
% memory must be initialized (so we need 'memory') ...
quit :- write('Quitting.'), nl, abort.
step :- step(1),!.
step(N): - for(1,N,X), execute_instruction, N == X, !.
status :- state(R,P,S,F), write(state(R,P,S,F)), n1.
my_trace :- my_trace(1).
my_trace(N):-for(1,N,X), execute_instruction, show(all), nl, N == X.
reset_processor :-
                retract(state(_,_,_,_)),
                asserta(state(regs(0,0,0,0,0,0,0),0,254,flags(0,0,0,0,0))).
change(Adr, Val) :-
                number(Adr),!,
                put_mem(Adr, Val).
change(Reg, Val) :-
                reg(Reg, X),!,
                retract(state(R,P,S,F)),
                argrep(R, X, Val, NewR),
                asserta(state(NewR,P,S,F)).
change(Flag, Val) :-
                fl(Flag, X),!,
                retract(state(R,P,S,F)),
                argrep(F, X, Val, NewF),
                asserta(state(R,P,S,NewF)).
change(pc, P) :-
                !, retract(state(R,_,S,F)),
                asserta(state(R,P,S,F)).
change(sp, S) :-
                !, retract(state(R,P,_,F)),
                asserta(state(R,P,S,F)).
change(bc, Val) :-
                !, state(R,_,_,_),
                arg(2,R,B), arg(3,R,C),
                ifthen(valid_adr(B,C), put_mem(B,C,Val)).
change(de, Val) :-
                !, state(R,_,_,_),
                arg(4,R,D), arg(5,R,E),
                ifthen(valid_adr(D,E), put_mem(D,E,Val)).
change(hl, Val) :-
               !, state(R,_,_,),
arg(6,R,H), arg(7,R,L),
               ifthen(valid_adr(H,L), put_mem(H,L,Val)).
show(Reg) :-
               reg(Reg, X),
               state(R,_,_,_),
               arg(X,R,T), dec_hex_byte(T,HT),
               tab(1), concat([Reg, '(', HT, ')'], Y), write(Y),!.
```

```
show(bc) :-
                state(R,_,_,_),
                arg(2,R,B), arg(3,R,C),
ifthenelse(valid_adr(B,C),
                             (get_mem(B,C,D), dec_hex_byte(D,DH)),
                             DH = $xx$),
                concat(['bc>',DH,'<'],Y),
                write(Y), !.
show(de) :-
                state(R,_,_,),
arg(4,R,D), arg(5,R,E),
                 ifthenelse(valid_adr(D,E),
                              (get_mem(D,E,D1), dec_hex_byte(D1,DH)),
                             DH = $xx$),
                concat(['de>',DH,'<'],Y),
                write(Y), !.
show(hl):-
                state(R,_,_,_),
                arg(6,R,H), arg(7,R,L),
                 ifthenelse(valid_adr(H,L),
                              (get_mem(H,L,D), dec_hex_byte(D,DH)),
                             DH = $xx$),
                concat(['hl>',DH,'<'],Y),
                write(Y), !.
show(pc) :-
                state(_,P,_,_), dec_hex_byte(P,PH),
                concat(['pc>',PH,'<'],Y),
                write(Y), !.
show(sp) :-
                state(_,_,S,_), dec_hex_byte(S,SH),
                concat(['sp>',SH,'<'],Y),
                write(Y), !.
show(regs) :-
                % show registers and pointers
                show(a), show(b), show(c), show(d), show(e), show(h), show(l),
                write(':'), show(bc), show(de), show(hl), show(pc), show(sp).
show(all) :-
                % show registers, pointers, and flags
                show(regs),
                nl, show(flags).
show(flags) :- % show flags only.
                state(_,_,,flags(Z,S,P,CY,AC)),
write('z = '),write(Z), write('; s = '), write(S),
                write('; p = '), write(P), write('; cy = '), write(CY),
                write('; ac = '), write(AC).
show(stack) :-
                state(_,_,S,_), S1 is S + 15,
                top_of_memory(TOM), min(S2,S1,TOM),
                show(S,S2).
show(Mem) :-
                % show a single memory location's contents.
                show(Mem, Mem),!.
show :-
                % starting with current PC address, show 16 bytes.
                state(_,P,_,_),
                top_of_memory(TOM), TOM_1 is TOM - 1,
                End is P + 15, min(Q, TOM_1, End),
                show(P,Q),!.
show(Lo, Hi) :- % show specified memory range.
                number(Lo), number(Hi),
                Diff is Hi - Lo + 1,
                show_mem(Diff, Lo),!.
show_mem(0, _) :- !.
```

STARTUP.C Contributed by Joel West. Accompanies the review "Macintosh C Compilers Revisited," August 1987, page 219.

```
/* Listing 9: Standard startup source */
/* startup.c */
#ifdef HYPERC
/* Won't create window for us */
        stdTerm(PStr("Hyper-C window"));
    EXTERN CHAR getKey();
#define getchar() getKey(TRUE)
#endif
#ifdef MPU68000
/* Aztec needs this */
pascal long TickCount() = 0xa975;
#endif
/* Start timing */
       long time;
        puts("Press any key to begin timed test: ");
        getchar();
        puts("\nStarting\n");
        time = TickCount();
```

MEMORY.ARI Contributed by Alex Lane. Accompanies the article "Simulating a Microprocessor," August 1987, page 161.

```
% File: MEMORY.ARI
% Note:
% Since this file is very repetitious, only a small portion of the
% file is represented here.
% You can generate a complete copy of the file by running the
   basic program MEMORY.BAS
% This file contains a predicate that initializes a 256-byte memory
% (shades of the mid-70s!).
% To "run" a program on the simulated microprocessor, insert the
%
   appropriate machine code bytes into the memory locations
init_mem :-
   asserta(top_of_memory(255)),
                             % NOP
   assertz(memory(0,0)),
                              % NOP
   assertz(memory(1,0)),
   assertz(memory(2,0)),
                              % NOP
```

```
assertz(memory(3,0)),
                                NOP
                             9/2
   assertz(memory(4,0)),
                             0/2
                                NOP
%
   assertz(memory(253,0)), %
                                NOP
   assertz(memory(254,0)),
                             %
                                NOP
   assertz(memory(255,0)).
                             %
                                NOP
% end
```

TOKENS80.ARI Contributed by Alex Lane. Accompanies the article "Simulating a Microprocessor," August 1987, page 161.

```
% Subject: TOKENS80.ARI - from Alex Lane: "Simulating a Microprocessor"
% This version of get_token_list(_) assumes all input numbers are in
  hexadecimal and delivers decimal numbers in the output; i.e.,
% entering 'show Of' results in [show, 15].
                                                         % read a sentence from the terminal
get_token_list(Result) :-
               read_line(0,String),
                                                         % read a line of input from the console
                list_text([Char Tail], String),
                tokenize([Char Tail], [], Result).
                                                         % if head of list starts with letter
tokenize([H|T],List,L):-
                letter(H, Letter),!,
                restword(T,[Letter],Word,Rem),
                                                         % get the rest of a word
                append(List, [Word], Nlist),
                tokenize(Rem, Nlist, L).
                                                         % recurse, tokenize rest of list
tokenize([H|T], List, L) :-
                                                         % if head of list starts with digit (0-9)
                digit(H),!,
                rest_num(T,[H],Num, Rem),
                                                         % get a number (Num will be decimal)
                append(List, [Num], Nlist),
                tokenize(Rem, Nlist, L).
                                                          % recurse, tokenize rest of list
tokenize([_|T], List, L) :-
                                                          % if head of list is not letter or digit,
                                                          % ignore it.
                !, tokenize(T,List,L).
                                                          % stop recursion.
tokenize([],List,List).
restword([HT], List, Word, X):-
                letter(H, Letter),!,
                append(List, [Letter], Nlist),
                restword( T, Nlist, Word, X).
restword([32|T], List, Word, T):-
                name (Word, List),!.
restword([_|T], List, Word, X):-
                !, restword( T, List, Word, X).
restword([], List, Word, []):-
                !, name(Word, List).
rest_num([H|T], List, Num, X):-
                hexdigit(H,_),!,
                                                          % rest of number may have 0-9, a-f
                append(List, [H], Nlist),
                rest_num(T, Nlist, Num, X).
                                                          % space finishes number, go convert.
rest_num([32|T], List, Num, T):-
                cname(Num, List, 0),!.
                                                          % nothing left, go convert.
rest_num([],List,Num,[]):-
                !, cname(Num, List, 0).
cname(F, [X []], N):-
                                                          % finished.
                !, hexdigit(X,Y), FisN+Y.
```

August

```
cname(Number, [H|T], In):-
                                                           % convert the input hex number to
                                                           % a decimal. User need never know!
                hexdigit(H,H1),
                H2 is (In + H1) * 16,
                                                           % recurse with shorter list in T
                cname (Number, T, H2).
letter(C, C) :-
                C>= 97, C =< 122, !.
                                                           % 97 is "a", 122 is "z"
letter(C, D) :-
                C>= 65, C =< 90, !,
                                                           %65 is "A", 90 is "Z"
                DisC+32.
                                                           % 32 is "a"-"A"
hexdigit(D,E) :-
                digit(D), E is D - 48.
hexdigit(D,F):-
                letter(D,E), E >= 97, E =< 102,
                FisE-87.
digit(C):-
                C >= 48, C = < 57.
                                                           % 48 is "0", 57 is "9".
% end: TOKENS80.ARI
```

MEMORY.BAS Contributed by Alex Lane. Accompanies the article "Simulating a Microprocessor," August 1987, page 161.

```
10 'Program to create MEMORY. ARI file for Prolog program
20' for "Simulating a Microprocessor"
30' by Alex Lane
40'
100 OPEN "MEMORY.ARI" FOR OUTPUT AS 1
110 PRINT#1, "% File: MEMORY.ARI"
120 PRINT#1, "% from Simulating a Microprocessor by Alex Lane"
130 PRINT#1, "%"
140 PRINT#1, "% This file contains a predicate that initializes"
150 PRINT#1, "% a 256-byte memory"
160 PRINT#1, "% To 'run' a program on the simulated microprocessor,
170 PRINT#1, "% insert the appropriate machine code bytes into
180 PRINT#1, "% the memory locations
190 PRINT#1, "%"
200 PRINT#1, "init_mem :-"
210 PRINT#1, " asserta(top_of_memory(255)),"
220 FOR I = 0 TO 254
230 IS$ = STR$(I)
240 PRINT#1, " assertz(memory("; IS$; ",0)), % NOP"
250 NEXT I
260 PRINT#1, " assertz(memory( 255,0)). % NOP"
300 PRINT#1, "%"
310 PRINT#1, "% end"
320 CLOSE
400 END
```

INTERFACE.C Contributed by Joel West. Accompanies the review "Macintosh C Compilers Revisited," August 1987, page 219.

```
/* Listing 6: Interface benchmark */
#define HYPERC
/* interface.c */
/* Modified by Joel West, April 14, 1987 for all systems */
#include <stdio.h>
#ifdef HYPERC
#include <TBXTypes.h>
```

```
#include <events.h>
#else
# ifdef MACC
  include <macdefs.h>
   include < events.h>
#else
# ifdef AZTEC
  include <window.h>
   include < event.h>
# else
  ifdef LSC
  include < EventMgr.h>
  endif
# endif
# endif
#endif
#define COUNT 10000
main()
  int i, eMask1, eMask2, bool1, bool2;
 EventRecord eRcrd1, eRcrd2;
#include "startup.c"
  eMask1 = eMask2 = -1;
  for (i = 0; i < COUNT; ++i)
            bool1 = GetNextEvent(eMask1, &eRcrd1);
             bool2 = GetNextEvent(eMask2, &eRcrd2);
             bool1 = GetNextEvent(eMask1, &eRcrd1);
             bool2 = GetNextEvent(eMask2, &eRcrd2);
             bool1 = GetNextEvent(eMask1, &eRcrd1);
             bool2 = GetNextEvent(eMask2, &eRcrd2);
             bool1 = GetNextEvent(eMask1, &eRcrd1);
             bool2 = GetNextEvent(eMask2, &eRcrd2);
#include "done.c"
```

START80.ARI Contributed by Alex Lane. Accompanies the article "Simulating a Microprocessor," August 1987, page 161.

```
% START80.ARI - from A. Lane: "Simulating an 8085 with Prolog"
% This file starts everything up.
go :-
  write('Loading memory.'), nl,
  consult(memory),
  write('Initializing memory.'),nl,
  init_mem.
  abolish(init_mem/0),
  write('Loading monitor.'), nl,
  consult(mon80),
  write('Loading command parser.'),nl,
  consult(parse80),
  write('Loading tokenizer.'), nl,
  consult(tokens80),
  write('Loading miscellaneous routines.'),nl,
  consult(preds80),
  write('Loading help information'), nl,
  consult(help80),
  write('Loading opcode predicates.'), nl,
  consult(ops8085),
  asserta(state(regs(0,0,0,0,0,0,0),0,255,flags(0,0,0,0,0))),
```

August

```
sign_on,
  monitor.
% op(_) :- write('instruction executed').
monitor :-
  repeat,
  prompt,
  get_token_list(Tokens),
  ifthenelse(parse_tokens(Command, Tokens,[]), % run command thru DCG. if valid
              call(Command),
                                                      % do it, else
              write('invalid request')),
                                                      % tell user no.
  fail.
execute_instruction :-
   retract(state(R,PC,SP,F)),
   NewPC is PC + 1,
   asserta(state(R, NewPC, SP, F)),
   get_mem(PC, Instruction),
   op(Instruction),!.
sign_on :-
  cls,
  write('8085 simulator'),nl,
  write($ Type 'h' for help.$), nl.
prompt :-
  tget(_,Col),
                     % find out what column cursor's in.
  ifthen(Col \= 0, nl), % if cursor not in column 0, new line.
  write('-<'),!.
                         % print prompt.
% end START80.ARI
```

SIEVE.C Contributed by Joel West. Accompanies the review "Macintosh C Compilers Revisited," August 1987, page 219.

```
/* Listing 1: Sieve benchmark */
/* sieve.c */
#define LSC
        Eratosthenes Sieve prime number program from BYTE, January 1983
        Modified by Joel West, April 13, 1987, for 16-bit short
#define REGISTER
#include <stdio.h>
#ifdef HYPERC
#include < TBXTypes.h>
#include <events.h>
#endif
#define TRUE
#define FALSE
                  0
#define size 8190
char flags [size + 1];
main()
        REGISTER short i, prime, k, count, iter;
#include "startup.c"
        for (iter = 1; iter <= 10; iter++)
                                                                                                               /* do program 10 times */
                   count = 0;
                                                                                                                     /* prime counter */
                    for (i = 0; i <= size; i++)
                                                                                                                /* set all flags true */
                             flags[i] = TRUE;
                    for (i = 0; i <= size; i++)
                             if (flags [i])
                                                                                                                      /* found a prime */
```

```
{
                                          prime = i + i + 3;
                                                                                                                     /* twice index + 3 */
                                          for (k = i + prime; k <= size; k+= prime) flags [k] = FALSE;
                                                                                                                  /* kill all multiple */
                                          count++;
                                                                                                                       /* primes found */
 #include "done.c"
SORT.C Contributed by Joel West. Accompanies the review "Macintosh C Compilers Revisited," August 1987, page 219.
 /* Listing 4: Sort benchmark */
 /* sort.c */
#define LSC
       sorting benchmark--calls randomly the number of times specified by MAXNUM to create an array of long integers, then does a
       quicksort on the array of longs. The program does this for the number of times specified by COUNT.
Modified by Joel West, April 13, 1987, for 16-bit short
#include <stdio.h>
#ifdef HYPERC
#include <TBXTypes.h>
#include <events.h>
#endif
#define REGISTER
#define MAXNUM 1000
#define COUNT 10
#define MODULUS ((long) 0x20000)
#define C 13849L
#define A 25173L
long seed = 7L;
long random();
long buffer [MAXNUM] = {0};
main()
             REGISTER short i, j;
             REGISTER long temp;
#include "startup.c"
             printf ("Filling array and sorting %d times\n", COUNT);
             for (i = 0; i < COUNT; ++i)
                          for (j = 0; j < MAXNUM; ++j)
                                    {
                                              temp = random (MODULUS);
                                              if (temp < OL)
                                                  temp = (-temp);
                                              buffer[j] = temp;
                                    printf ("Buffer full, iteration %d\n", i);
                                    quick (0, MAXNUM - 1, buffer);
#include "done.c"
```

quick (lo, hi, base) REGISTER short lo, hi;

```
long base [];
             REGISTER int i, j;
             REGISTER long pivot, temp;
             if (lo < hi)
                           for (i = lo, j = hi-1, pivot = base [hi]; i < j;)
                                     while (i < hi && base [i] <= pivot) .
                                        ++1;
                                     while (j > lo && base [j] >= pivot)
                                        --j;
                                     if (i < j)
                                         temp = base [i];
                                        base [i] = base [j];
                                        base [j] = temp;
                           temp = base [i];
                           base [i] = base [hi];
                           base [hi] = temp;
                           quick (lo, i-1, base);
                           quick (i + 1, hi, base);
long random (size)
REGISTER long size;
             seed = seed * A + C;
             return (seed % size);
```

SAVAGE.C Contributed by Joel West. Accompanies the review "Macintosh C Compilers Revisited," August 1987, page 219.

```
/* Listing 2: Savage benchmark */
/* savage.c */
#define LSC
** savage.c -- floating-point speed and accuracy test. C version
** derived from BASIC version that appeared in Dr. Dobb's Journal,
** Sept. 1983, pp. 120-122.
Modified by Joel West, April 14, 1987
For accuracy on the Macintosh, we want to use the SANE 80-bit extended type for all compilers. This is:
         Lightspeed C double
                        extended
         Mac C
         Hyper C
                        extended
         Aztec C
                        N/A
#define ILOOP 2500
#include <stdio.h>
#ifdef MACC
#define EXTENDED extended
#include <sane.h>
#else
#ifdef HYPERC
#define EXTENDED extended
#include <TBXTypes.h>
#include <events.h>
#else
#define EXTENDED double
#endif
```

#endif

FLOAT.C Contributed by Joel West. Accompanies the review "Macintosh C Compilers Revisited," August 1987, page 219.

```
/* Listing 7: Float benchmark */
#define LSC
/* float.c */
/* simple benchmark for testing floating-point speed of c libraries does repeated multiplications and divisions in a loop that is
   large enough to make the looping time insignificant */
#include <stdio.h>
#ifdef MACC
#define EXTENDED extended
#else
#ifdef HYPERC
#define EXTENDED extended
#include <TBXTypes.h>
#include <events.h>
#else
#define EXTENDED double
#endif
#endif
#define CONST1 3.141597E0
#define CONST2 1.7839032E4
#define COUNT 10000
main()
        EXTENDED a, b, c;
        int i;
#include "startup.c"
         a = CONST1:
         b = CONST2;
         for (i = 0; i < COUNT; ++i)
               c = a * b;
                c = c / a;
                c = a * b;
                c = c / a;
                c = a * b;
                c = c / a;
                c = a * b;
                c = c / a;
                c = a * b;
```

```
c = c / a;
c = a * b;
c = c / a;
c = a * b;
c = c / a;
}
#include "done.c"
```

DONE.C Contributed by Joel West. Accompanies the review "Macintosh C Compilers Revisited," August 1987, page 219.

```
/* Listing 10: standard termination header */
/* done.c */
/* End timing */
    time = TickCount() - time;
    printf("ticks=%ld\n", time);
    printf("Press any key to return to FINDER: ");
    getchar();
```

FIB.C Contributed by Joel West. Accompanies the review "Macintosh C Compilers Revisited," August 1987, page 219.

```
/* Listing 5: Fib benchmark */
#define LSC
/* Fib.c */
/* Fibonacci benchmark, modified by Joel West, 4/13/87 */
#include <stdio.h>
#ifdef HYPERC
#include <TBXTypes.h>
#include <events.h>
#endif
#define REGISTER
#define NTIMES 10 /* number of times to compute Fibonacci value */
#define NUMBER 24 /* biggest one we can compute with 16 bits */
            /* compute Fibonacci value */
main()
             REGISTER short i;
             REGISTER unsigned short value;
             unsigned short fib();
#include "startup.c"
             for (i = 1; i <= NTIMES; i++)
                value = fib(NUMBER);
#include "done.c"
   exit(0);
unsigned short fib(x)
                          /* compute Fibonacci number recursively */
  REGISTER short x;
             if (x > 2)
```

```
return (fib(x-1) + fib(x-2));
            else
               return (1);
}
```

FILEIO.C Contributed by Joel West. Accompanies the review "Macintosh C Compilers Revisited," August 1987, page 219.

```
/* Listing 8: Fileio benchmark */
#define LSC
/* fileio.c */
       file reading and writing benchmark sequentially writes a 65,000-byte file on disk, generates random long numbers, and uses
       these modulo 65,000 to read and write strings of ODDNUM bytes with the file-handling system of the c package; the random-number
       generator is set to a specific seed, so that all compilers should generate the same code
  Fixed by Joel West, April 16, 1987, to use UNIX-standard creat()
  and open() parameters.
#include <stdio.h>
#ifdef LSC
#include <unix.h>
#else
#ifndef MACC
                                 /* this is the right way to do it */
#include <fcntl.h>
#endif
#endif
#ifndef MACC
#define FILEMODE 0666
                                 /* the normal rw-rw-rw */
#else
                                 /* Mac C only */
#define FILEMODE 0x7
#define O_RDONLY O
#define O_WRONLY 1
#define O_RDWR 2
#endif
#ifdef LSC
#define abort AbOrT
                                 /* Lightspeed C has an 'abort' entry point */
#endif
#define ERROR -1
#define READERR O
#define OKCLOSE O
/* For lseek() */
#define BEG 0
#define CURR 1
#define END 2
#define FILESIZE 65000L
#define COUNT 500
#define C 13849L
#define A 25173L
#define ODDNUM 23
long seed = 7L;
long random (), lseek ();
main ()
{
             int i;
             long j, pos;
             int fd;
```

char buffer [ODDNUM + 1];

August

#include "startup.c"

```
if ((fd = creat ("test.dat", FILEMODE)) == ERROR)
             abort ("Can't create data file\n");
             printf("File opened for sequential writing\n");
            for (j = 0; j < FILESIZE; ++j)
             if (write(fd, "x", 1) == ERROR)
              abort ("Unexpected EOF in writing data file\n");
            if (close (fd) != OKCLOSE)
             abort ("Error closing data file\n");
            else
             printf ("Normal termination writing data file\n");
            if ((fd = open ("test.dat", O_RDWR)) == ERROR)
             abort ("Can't open data file for random reading and writing\n");
            else
              printf ("File opened for random reading and writing\n");
            for (i = 0; i < COUNT; ++i)
                      j = random (FILESIZE);
                      if (j < OL)
                       j = (-j);
                      if (FILESIZE - j < ODDNUM)
                       continue;
                      if ((pos = lseek (fd, j, BEG)) == -1L)
                        abort ("Error reading at random offset\n");
                      if (read (fd, buffer, ODDNUM) == READERR)
                       abort ("Error reading at random offset\n");
                      j = random (FILESIZE);
                      if (j < OL)
                        j = (-j);
                       if (FILESIZE - j < ODDNUM)
                        continue;
                       if ((pos = lseek (fd, j, BEG)) == -1L)
                        abort ("Error seeking to random offset\n");
                       if (write (fd, buffer, ODDNUM) == READERR)
                        abort ("Error writing at random offset\n");
             if (close (fd) != OKCLOSE)
                       abort ("Error closing data file\n");
             else
                       printf ("Normal termination from random reading and writing\n");
#include "done.c"
long random (size)
long size;
             seed = seed * A + C;
             return (seed % size);
}
abort (message)
char *message;
             printf (message);
             exit (ERROR);
```

CHAOSBEN.BAS Contributed by Jerry Pournelle. Accompanies "Computing At Chaos Manor: Faster, Bigger, Better," August 1987, page 243.

```
The Benchmark Program
REM A benchmark program to test machines, compilers, and languages.
REM ** DECLARATIONS
DEFINT I - N
DEFINT E
DEFDBL A - C
DEFDBL S
REM Variable "start$" is a string.
REM *** CONSTANTS
ELEMENTS = 50
SUM = 0
BELL$ = CHR$(7)
REM *** DIMENSIONS
DIM A(Elements, Elements)
DIM B(Elements, Elements)
DIM C(Elements, Elements)
REM *** PROGRAM
                   'Clear the screen or it'll scroll funny
CLS
Print "This is a program to fill two matrices of ";
print Elements;
print "elements, multiply them, and sum the result."
print
print "It can be used as a benchmark program to test";
print "languages, compilers, or machines."
print
print "Written by Jerry Pournelle, May 1987."
print "Not copyrighted."
input "Start timer. Enter any character to begin. "; start$
GOSUB FillA
Print "A Filled."
GOSUB FillB
Print "B Filled"
GOSUB FillC ' Needed because some compilers can't cope.
Print "C Filled"
GOSUB DoMultiply
Print "Multiplied"
GOSUB SumItUp
Print "Sum = "; Sum
BEEP (5)
END
REM ******** PROCEDURES *******
FillA:
  FOR i = 1 to Elements
    FOR j = 1 to Elements
          A(i,j) = i + j
        NEXT
      NEXT
RETURN ' End FillA
```

August

```
FillB:
 FOR i = 1 to Elements
       FOR j = 1 to Elements
        B(i,j) = (i+j) / j
        NEXT
     NEXT
RETURN 'End FillB
FillC:
      FOR I = i = 1 to Elements
        FOR j = 1 to Elements
         C(i,j) = 0
        NEXT
      NEXT
RETURN ' End FillC
DoMultiply:
      FOR i = 1 to Elements
        FOR j = 1 to Elements
          FOR k = 1 to Elements
           C(i,j) = C(i,j) + A(i,k) * B(k,j)
          NEXT
        NEXT
      NEXT
RETURN ' End DoMultiply
SumItUp:
      FOR i = 1 to Elements
        FOR j = 1 to Elements
         SUM = SUM + C(i,j)
        NEXT
      NEXT
RETURN ' End SumItUp
Print "ESAD!" 'Shouldn't be able to get here.
```

ALSPRO.TST Contributed by Alex Lane. From "ALS Prolog," BYTE, September 1987.

```
"Translations" of Prolog benchmarks first used on Turbo Prolog.
%
% When running these benchmarks, remember that the read/1 predicate expects to see a period to signal the end of input!
% Note: All errors from the original set of benchmarks have been fixed.
                                            a.lane (4/17/87)
% Factorial Benchmark Test
fact :-
  write('Enter number of iterations'),
  read(Iter), nl,
  write('Enter factorial number'),
  read(Num), nl, nl,
  time(Start),
  repeat(Iter, Num),
  time(Finish), nl, Overall is Finish - Start,
  write('Time is'), write(Overall), nl.
  factorial(1,1) :- !.
  factorial(N, Result) :-
    N1 is N - 1,
    factorial(N1, Temporary),
    Result is N * Temporary.
  repeat(0,R) :- write(X),nl.
  repeat(N,R) :- factorial(R,_),
               N1 is N - 1,
               repeat(N1,R).
  time(Time) :- Time is cputime.
% List-Reversal Test Program
lrev :-
  write('Enter cycle length'),
  read(N),
  time(Start),
  cycle(N),
  time(Finish), Overall is Finish - Start,
  write('Time = '), write(Overall), nl.
append([], L, L).
append([Z|L1], L2, [Z|L3]):-append(L1, L2, L3).
37,38,39,40,41,42,43,44,45,46,47,48,49,50], L).
lipshort(L):-rev([1,2,3,4,5,6,7,8,9,10], L).
 rev([],[]).
rev([H|T], L):-rev(T,Z), append(Z,[H], L).
 cycle(N) := N1 is N - 1, lips(_), cycle(N1).
 time(Time) :- Time is cputime.
```

```
Floating-Point Test Program
% This floating-point benchmark is significantly different from the one performed on Turbo Prolog.
% The heart of the Turbo benchmark reads:
%
   calc(A,B):-
        Cis 1.0,
%
        C1 is C * A,
%
        C2 is C1 * B,
%
        C3 is C2 / A,
        C is C3 / B,
%
        bound(C).
% which is flawed for two reasons. First, the value of C is reinitialized each time the predicate calc/2 is called, which defeats
% one of the reasons for performing these operations 5000 times: to see if there is any cumulative error. Second, the Turbo
% benchmark works only if the result of C3 / B is *exactly* 1.0. If there is any error (i.e., if, as is true with ALS Prolog,
\% C3 / B is 1.000000000000000000001) the thing won't fly at all.
float :-
  time(Start),
  cycle(5000, 1.0, 2.71828, 3.14159),
  time(Finish), Overall is Finish - Start,
  write('Time is '), write(Overall), nl.
  calc(In,Out,A,B) :-
     C1 is In * A,
     C2 is C1 * B,
     C3 is C2 / A,
     Out is C3 / B.
  cycle(0,C,A,B):-
    write('Cis',C),nl.
  cycle(N,C,A,B):-
     calc(C,CF,A,B),
     N1 is N - 1,
     cycle(N1,CF,A,B).
time(Time) :- Time is cputime.
% Math Functions Test Program
 write('Doing square root...'),nl,
 time(T1),
 cyclesqrt(1000,_,T1),
 write('Doing logs..'),nl,
 time(T2),
 cycleln(1000,_,T2),
 write('Doing exp..'),nl,
 time(T3),
 cycleexp(1000,_,T3),
 write('Doing atan..'), nl,
 time(T4),
 cycleatan(1000,_,T4),
 write('Doing sin...'), nl,
 time(T5),
 cyclesin(1000,_,T5).
cyclesqrt(0,R,T1):-time(T6), Stime is T6-T1,
                  write('SQRT:'), write(Stime), n1,!.
cyclesqrt(N, R, T) :-
    N > 0, N1 is N - 1, R is sqrt(100.0), cyclesqrt(N1,R,T).
cycleln(0,R,T2):-time(T7), Ltime is T7-T2,
                write('LN:'), write(Ltime), nl.
cycleln(N, R,T) :-
    N > 0, N1 is N - 1, R is log(100.0), cycleln(N1,R,T).
```

```
cycleexp(0,R,T3):-time(T8), Etime is T8-T3,
                 write('EXP: '), write(Etime), nl.
cycleexp(N, R,T) :-
    N > 0, N1 is N - 1, R is exp(10.0), cycleexp(N1,R,T).
cycleatan(0,R,T4):-time(T9), Atime is T9-T4,
                  write('ATAN:'), write(Atime), nl.
cycleatan(N, R,T) :-
    N > 0, N1 is N-1, R is atan(10.0), cycleatan(N1, R, T).
cyclesin(0,R,T5): - time(T10), Stime is T10 - T5,
                 write('SIN:'), write(Stime), nl.
cyclesin(N, R,T) :-
    N > 0, N1 is N-1, R is sin(10.0), cyclesin(N1, R, T).
time(Time) :- Time is cputime.
% Disk Read Program
dread :-
  see('a:tempo.dat'),
  time(Start),
  get_text(512),
  time(Finish), Overall is Finish - Start,
  seen, see(user),
write('Time = '), write(Overall), nl,
  write('DONE'), nl.
get_text(0).
get_text(N):-
     read(Str),
     N1 is N-1,
     get_text(N1).
% Disk Write Benchmark
dwrite :-
  tell('a:tempo.dat'),
  time(Start),
  send_text(512),
  time(Finish), Overall is Finish - Start,
  told, tell(user),
  write('Time = '), write(Overall), nl,
  write('DONE'), nl.
send_text(0).
send_text(N) :-
      write('x2345678123456781234567812345670123456781234567812345678123456701234
56781234567812345678123456701234567812345678123456781234567.'),
     nl, N1 is N - 1,
      send_text(N1).
time(Time) :- Time is cputime.
% Tower of Hanoi Test Program
hanoi :-
   write('Enter tower height'),
   read(High),
   time(Start),
   hanoi(High),
   time(Finish), Overall is Finish - Start,
   write('Time:'), write(Overall), nl.
 hanoi(N): - move(N, left, center, right).
```

```
move(0, _, _, _) :-!.
 move(N, A, B, C) :-
      M is N-1,
      move(M, A, C, B),
      move2(bottom, A, B),
      move(M, C, B, A).
 move2(bottom, A, B) :-
      write('Move the disk on '),
      write(A),
      write(' to'),
      write(B),
      nl.
 time(Time) :- Time is cputime.
 %-----
 % Sieve Test Program
 eratosthenes :-
   time(Start),
   cycle(10),
   time(Finish), Overall is Finish - Start,
   write('Time is'), write(Overall), nl.
primes(Limit, Ps):-
        integers(2, Limit, Is),
        sift(Is, Ps).
integers (Low, High, [Low Rest]) :-
         Low = < High, !, M is Low + 1,
         integers (M, High, Rest).
integers(_,_,[]).
sift([],[]).
sift([I|Is],[I|Ps]):-
        remove(I, Is, New),
        sift( New, Ps ).
remove(_,[],[]).
remove(P,[I|Is],[I|Nis]):-
I mod P = \ = 0,!,
        remove(P, Is, Nis).
remove(P,[I|Is],Nis):-
        I mod P = : = 0,
        remove(P, Is, Nis).
cycle(0).
cycle(N) :-
        N1 is N - 1,
        primes(100,_),
        cycle(N1).
time(Time) :- Time is cputime.
 This program is called with the query "?-boresea(X)."
  X is the number of loop iterations executed. It should be big enough to give significant results.
  suggested value for X: 100 for interpreted code
                        1000 for compiled code
   average values for C-prolog interpreter:
%
         X=1000, Tloop=27.1 T.comp=1.0 Tnet=26.1 Klips=7.7
boresea(X)
     :- T1 is cputime,
       do_max_KLips(X),
                                          % calls the loop to execute the
       T2 is cputime,
                                          % sequence of 200 predicates
       compens_loop(X),
                                          % compensation loop
       T3 is cputime,
       print_times(T1, T2, T3, X, 200).
                                          % compute and print results
compens_loop(0).
                                                % compensation loop
compens_loop(X):-Yis X-1, compens_loop(Y).
```

```
print_times(T1,T2,T3,X,I):-
                                             % prints the results
         TT1 is T2 - T1,
         TT2 is T3 - T2,
         TT is TT1 - TT2,
        write('T overall loop:
                                            '), write(TT1), nl,
         write('T compens loop:
                                            '), write(TT2), nl,
         write('Tnet:
                                            '), write(TT), nl,
         write('KLips:
         Li is I * X,
         Lips is Li / TT,
         KLips is Lips / 1000,
         write(KLips), nl, nl.
 do_max_KLips(0).
                                               % loop calling the actual benchmark
 do_max_KLips(X):-lips1, Y is X-1, do_max_KLips(Y).
 % predicates to test call
lips1:-lips2.
lips2:-lips3.
 lips3:-lips4.
lips4:-lips5.
lips5 :- lips6.
lips6:-lips7.
lips7:-lips8.
lips8:-lips9.
lips9:-lips10.
lips10:-lips11.
lips11:-lips12.
lips12:-lips13.
lips13:-lips14.
lips14:-lips15.
lips15:-lips16.
lips16:-lips17.
lips17:-lips18.
lips18:-lips19.
lips19:-lips20.
lips20 :- lips21.
lips21:-lips22.
lips22:-lips23.
lips23:-lips24.
lips24:-lips25.
lips25 :- lips26.
lips26:-lips27.
lips27:-lips28.
lips28:-lips29.
lips29:-lips30.
lips30 :- lips31.
lips31:-lips32.
lips32:-lips33.
lips33:-lips34.
lips34:-lips35.
lips35 :- lips36.
lips36:-lips37.
lips37:-lips38.
lips38:-lips39.
lips39:-lips40.
lips40 :- lips41.
lips41:-lips42.
lips42:-lips43.
lips43:-lips44.
lips44:-lips45.
lips45 :- lips46.
lips46 :- lips47.
lips47:-lips48.
lips48:-lips49.
lips49:-lips50.
lips50 :- lips51.
lips51:-lips52.
lips52:-lips53.
lips53:-lips54.
lips54:-lips55.
lips55 :- lips56.
lips56:-lips57.
lips57:-lips58.
```

```
lips58 :- lips59.
lips59:-lips60.
 lips60 :- lips61.
lips61:-lips62.
lips62:-lips63.
lips63:-lips64.
lips64:-lips65.
lips65:-lips66.
lips66:-lips67.
lips67:-lips68.
lips68:-lips69.
lips69:-lips70.
lips70 :- lips71.
lips71:-lips72.
lips72:-lips73.
lips73:-lips74.
lips74:-lips75.
lips75 :- lips76.
lips76:-lips77.
lips77:-lips78.
lips78:-lips79.
lips79:-lips80.
lips80:-lips81.
lips81:-lips82.
lips82:-lips83.
lips83:-lips84.
lips84:-lips85.
lips85 :- lips86.
lips86:-lips87.
lips87:-lips88.
lips88:-lips89.
lips89:-lips90.
lips90:-lips91.
lips91:-lips92.
lips92:-lips93.
lips93:-lips94.
lips94:-lips95.
lips95:-lips96.
lips96:-lips97.
lips97:-lips98.
lips98:-lips99.
lips99:-lips100.
lips100:-lips101.
lips101:-lips102.
lips102:-lips103.
lips103:-lips104.
lips104:-lips105.
lips105:-lips106.
lips106:-lips107.
lips107:-lips108.
lips108:-lips109.
lips109:-lips110.
lips110 :- lips111.
lips111:-lips112.
lips112:-lips113.
lips113:-lips114.
lips114:-lips115.
lips115 :- lips116.
lips116:-lips117.
lips117:-lips118.
lips118 :- lips119.
lips119:-lips120.
lips120 :- lips121.
lips121:-lips122.
lips122 :- lips123.
lips123:-lips124.
lips124 :- lips125.
lips125 :- lips126.
lips126 :- lips127.
lips127:-lips128.
lips128 :- lips129.
lips129:-lips130.
lips130:-lips131.
lips131:-lips132.
lips132:-lips133.
lips133:-lips134.
```

```
lips134:-lips135.
lips135:-lips136.
lips136:-lips137.
lips137:-lips138.
lips138:-lips139.
lips139:-lips140.
lips140:-lips141.
lips141:-lips142.
lips142:-lips143.
lips143:-lips144.
lips144:-lips145.
lips145 :- lips146.
lips146:-lips147.
lips147:-lips148.
lips148:-lips149.
lips149:-lips150.
lips150:-lips151.
lips151:-lips152.
lips152:-lips153.
lips153:-lips154.
lips154:-lips155.
lips155:-lips156.
lips156:-lips157.
lips157:-lips158.
lips158:-lips159.
lips159:-lips160.
lips160 :- lips161.
lips161:-lips162.
lips162:-lips163.
lips163:-lips164.
lips164:-lips165.
lips165:-lips166.
lips166:-lips167.
lips167:-lips168.
lips168:-lips169.
lips169:-lips170.
lips170:-lips171.
lips171:-lips172.
lips172:-lips173.
lips173:-lips174.
lips174:-lips175.
lips175 :- lips176.
lips176:-lips177.
lips177:-lips178.
lips178:-lips179.
lips179:-lips180.
lips180:-lips181.
lips181:-lips182.
lips182:-lips183.
lips183:-lips184.
lips184:-lips185.
lips185 :- lips186.
lips186:-lips187.
lips187:-lips188.
lips188:-lips189.
lips189:-lips190.
lips190:-lips191.
lips191:-lips192.
lips192:-lips193.
lips193:-lips194.
lips194:-lips195.
lips195 :- lips196.
lips196:-lips197.
lips197:-lips198.
lips198:-lips199.
lips199:-lips200.
lips200.
% Choice Point Benchmark.
```

% The predicates are called:

% o "choice_point(N)" - creation of choice points

% N is the number of loop iterations executed

```
% predicate to test creation of choice points without
% backtracking
choice_point(N) :- T1 is cputime,
         cre_CP(N), T2 is cputime,
         compens_loop(N), T3 is cputime,
         print_times(T1,T2,T3,N,20).
% compensation loop, used to measure the time spent in
% the loop
compens_loop(0).
compens_loop(X):-Y is X-1, compens_loop(Y).
% loop to test choice point creation
cre CP(0).
cre_CP(N):-M is N-1, ccp1(0,0,0), cre_CP(M).
cre_CPOar(0).
cre_CPOar(N):-M is N-1, ccp1, cre_CPOar(M).
print_times(T1,T2,T3,X,I):-
                                              % prints the results
        TT1 is T2 - T1,
        TT2 is T3 - T2,
        TT is TT1 - TT2,
        write('T overall loop: write('T compens loop:
                                              '), write(TT1), nl,
                                             '), write(TT2), nl,
        write('T net:
                                             '), write(TT), nl,
        write('KLips:
        Li is I * X,
        Lips is Li / TT.
        KLips is Lips / 1000,
        write(KLips), nl, nl.
% ccpl creates 20 choice points
% ccpl is the beginning of a set of predicates composed of 2 clauses each. Every invocation of nd0 will create
% a sequence of 20 choice points. The body of the clauses are limited to one goal, thus avoiding a creation of environment
% when the clause is activated. ndO, and its successors, have three arguments to comply with our average static analysis
% results made on more than 30 real Prolog programs.
% ccpXX exists with 3 arguments, and 0 args.
ccp1(X,Y,Z):-ccp2(X,Y,Z).
cep1(X,Y,Z).
ccp2(X,Y,Z):-ccp3(X,Y,Z).
cep2(X,Y,Z).
ccp3(X,Y,Z):-ccp4(X,Y,Z).
ccp3(X,Y,Z).
cep4(X,Y,Z):-cep5(X,Y,Z).
cep4(X,Y,Z).
ccp5(X,Y,Z):-ccp6(X,Y,Z).
ccp5(X,Y,Z).
ccp6(X,Y,Z):-ccp7(X,Y,Z).
ccp6(X,Y,Z).
ccp7(X,Y,Z):-ccp8(X,Y,Z).
ccp7(X,Y,Z).
cep8(X,Y,Z):-eep9(X,Y,Z).
ccp8(X,Y,Z).
ccp9(X,Y,Z):-ccp10(X,Y,Z).
ccp9(X,Y,Z).
ccp10(X,Y,Z):-ccp11(X,Y,Z).
ccp10(X,Y,Z).
ccp11(X,Y,Z):-ccp12(X,Y,Z).
ccp11(X,Y,Z).
ccp12(X,Y,Z):-ccp13(X,Y,Z).
ccp12(X,Y,Z).
ccp13(X,Y,Z):-ccp14(X,Y,Z).
ccp13(X,Y,Z).
cep14(X,Y,Z):-cep15(X,Y,Z).
ccp14(X,Y,Z).
ccp15(X,Y,Z):-ccp16(X,Y,Z).
cep15(X,Y,Z).
cep16(X,Y,Z):-cep17(X,Y,Z).
ccp16(X,Y,Z).
```

ccp17(X,Y,Z):-ccp18(X,Y,Z).

ccp17(X,Y,Z).

```
ccp18(X,Y,Z):-ccp19(X,Y,Z).
ccp18(X,Y,Z).
cep19(X,Y,Z):-cep20(X,Y,Z).
ccp19(X,Y,Z).
ccp20(X,Y,Z).
cep20(X,Y,Z).
```

KAREX3.BAS Accompanies the article "Karmarkar's Algorithm" by Andrew M. Rockett and John C. Stevenson, BYTE, September 1987.

```
101 '
102 '
          KAREX3.BAS is a Microsoft BASIC Release 5 program
103 '
                  that solves EXAMPLE 3 of the article
104 1
105 '
                         KARMARKAR'S ALGORITHM
106 '
107 '
              by Andrew M. Rockett and John C. Stevenson
108 '
109 '
             This program was written by Andrew M. Rockett
110 '
111 '-
200 '
202 ' N is the number of unknowns and K is the number of equations
204 1
206 N = 8 : K = 4
208 '
210 K1 = K + 1 : K2 = 2*K1
212 DIM AO(N), XOLD(N), XNEW(N), CC(N), CP(N), A(K,N), B(K1,N), B1(K1,K2), B2(N,K1), B3(N,N)
214 \text{ FOR C} = 1 \text{ TO N} : AO(C) = 1/N : XNEW(C) = AO(C) :
     NEXT C
216 1
218 ' T is the tolerance
220 '
222 T = .001
224 1
226 ' ALPHA is usually set equal to 1/4 ...
228 '
230 ALPHA = .25
232 '
234 ITERATION = 0
236 1
238 ' Data for constraint matrix A
240 1
242 DATA 1, 0, -1, 0, 0, 0, 3, -3
244 DATA 1, 0, 0, 1, 0, 0, 0, -2
246 DATA 0, 1, 0, 0, 1, 0, 3, -5
248 DATA 0, 1, 0, 0, 0, -1, 4, -4
250 '
252 FOR R = 1 TO K : FOR C = 1 TO N : READ A(R,C) :
     NEXT C: NEXT R
254 1
256 ' Data for objective function CC
258 1
260 DATA 0, 0, 0, 0, 0, 0, 1, 0
262 1
264 FOR C = 1 TO N : READ CC(C) : NEXT C
266 1
268 \text{ V} = 0 : \text{FOR C=1 TO N} : \text{V} = \text{V} + \text{CC(C)*AO(C)} :
     NEXT C : VNEW = V
270 '
272 ' Main iteration process is the same as in KAREX1.BAS ...
274 1
300 WHILE VNEW/V > T
301 PRINT USING "###"; ITERATION; :
     FOR C=1 TO N: PRINT USING "###.###"; XNEW(C); :
     NEXT C : PRINT USING "####.######"; VNEW/V
302 ITERATION = ITERATION + 1
303 FOR C = 1 TO N : XOLD(C) = XNEW(C) : NEXT C
```

```
304 FOR R=1 TO K: FOR C=1 TO N:B(R,C)=A(R,C)*XOLD(C):
     NEXT C: NEXT R
305 FOR C=1 TO N:B(K1,C)=1:NEXT C
306 FOR R=1 TO K1 : FOR C=1 TO K2 : B1(R,C)=0 :
     NEXT C: NEXT R
307 FOR R=1 TO N : FOR C=1 TO K1 : B2(R,C)=0 :
     NEXT C: NEXT R
308 FOR R=1 TO N : FOR C=1 TO N : B3(R,C)=0 :
     NEXT C: NEXT R
309 FOR C=1 TO N : CP(C) = 0 : NEXT C
310 FOR R=1 TO K1: FOR C=1 TO K1:
        FOR I=1 TO N:B1(R,C)=B1(R,C)+B(R,I)*B(C,I):
         NEXT I:
     NEXT C: NEXT R
311 FOR I = 1 TO K1 : B1(I,I+K1)=1 : NEXT I
312 FOR R = 1 TO K1
313
      IF B1(R,R) <> 0 THEN 318
314
        I = R + 1
315
        IF I > K1 THEN PRINT "Error! BBT is SINGULAR!" :
         GOTO 405
316
        IF B1(I,R) = 0 THEN I = I+1 : GOTO 315
317
        FOR C = 1 TO K2 : SWAP B1(R,C), B1(I,C) : NEXT C
318 FOR I = R+1 TO K1:Z = B1(I,R)/B1(R,R):
       FOR C=1 TO K2:B1(I,C)=B1(I,C)-Z*B1(R,C):NEXT C:
      NEXT I
319 NEXT R
320 FOR R=K1 TO 2 STEP -1: FOR I = R-1 TO 1 STEP -1: Z = B1(I,R)/B1(R,R):
     FOR C=R TO K2:B1(I,C)=B1(I,C)-Z*B1(R,C):NEXT C:
     NEXT I:NEXT R
321 FOR R=1 TO K1:Z = B1(R,R):
        FOR C=1 TO K2:B1(R,C)=B1(R,C)/Z:NEXT C:
     NEXT R
322 FOR R=1 TO N: FOR C=1 TO K1:
       FOR J=1 TO K1:B2(R,C)=B2(R,C)+B(J,R)*B1(J,C+K1):
       NEXT J:
     NEXT C: NEXT R
323 FOR R=1 TO N: FOR C=1 TO N:
      FOR J=1 TO K1:B3(R,C)=B3(R,C)+B2(R,J)*B(J,C):
         NEXT J:
     NEXT C: NEXT R
324 \text{ FOR R} = 1 \text{ TO N} : B3(R,R) = B3(R,R) - 1 : NEXT R
325 FOR R=1 TO N: FOR C=1 TO N:B3(R,C)=-1*B3(R,C):
     NEXT C: NEXT R
326 FOR R=1 TO N: FOR C=1 TO N:B3(R,C)=B3(R,C)*XOLD(C):
     NEXT C: NEXT R
327 FOR R=1 TO N: FOR C=1 TO N: CP(R)=CP(R)+B3(R,C)*CC(C):
     NEXT C: NEXT R
328 AA=0:FOR C=1 TO N: AA = AA + CP(C)*CP(C): NEXT C
329 \text{ AA} = \text{SQR}(\text{AA}) : \text{FOR C=1 TO N} : \text{CP(C)} = \text{CP(C)}/\text{AA} :
     NEXT C
330 AA = SQR(N*(N-1))/ALPHA
331 FOR C=1 TO N : XNEW(C) = (AO(C) - CP(C)/AA)*XOLD(C) :
     NEXT C
332 AA=0: FOR C=1 TO N: AA = AA + XNEW(C): NEXT C
333 FOR C=1 TO N : XNEW(C) = XNEW(C)/AA : NEXT C
334 VNEW=0: FOR C=1 TO N: VNEW=VNEW+CC(C) *XNEW(C): NEXT C
335 1
336 ' FAILURE DETECTION routine based on equation (6) ...
337 1
338 ' You may wish to put this routine into KAREX1 and
     KAREX2 to
339 'observe the values appearing in (6) during the solutions
340 ' of EXAMPLES 1 and 2.
341 '
342 AA = 0
343 FOR C = 1 TO N
344 IF XNEW(C) > O THEN AA = AA + LOG(XNEW(C))
346 PRINT, LOG(VNEW/V), LOG(N) + AA/N - ITERATION/(8*N)
347 1
348 IF LOG(VNEW/V) > LOG(N) + AA/N - ITERATION/(8*N)
     THEN 400
349 1
350 WEND
351 '
```

```
400 PRINT : PRINT "Failure condition has occurred." :
    PRINT
401 PRINT USING "###"; ITERATION; :
    FOR C=1 TO N: PRINT USING "###.###"; XNEW(C); :
    NEXT C : PRINT USING "####.#####"; VNEW/V
402 1
403 PRINT: FOR C=1 TO N-2: PRINT XNEW(C)/XNEW(N),:
    NEXT C: PRINT
404 1
405 END
```

BAM.PAS From "Constructing an Associative Memory" by Bart Kosco, BYTE, September 1987.

```
program bam:
  { for further information
                            Rod Taber
                            General Dynamics
                            Electronics Division Mail Zone 7202-K
                            Box 85310
                            San Diego, CA 92138
 Mail without the Mail Zone takes 3 months.}
 {$R+, V+, K+, C-, U-}
 const
   maxrows
                    = 12;
   maxcolumns
                    = 12:
   maxentries
                    = 144;
   maxpatterns
                    = 4;
   screenrows
                    = 24;
   screencolumns
                    = 80:
 type
   threeD =
                     array[0..maxpatterns,1..1,1..maxentries] of integer;
   twoD =
                     array[1..maxentries] of integer;
   oneD =
                     array[1..maxentries] of integer;
   square =
                     array[1..maxentries,1..maxentries] of integer;
   Textin =
                    string[15];
   Ham:array[1..maxpatterns] of integer;
  Bipolar_A, Bipolar_B, Pattern_A, Pattern_B
                                                             :threeD;
   OriginalTestPattern,OutPatt
                                                             :oneD;
  TestPattern, A_Check, B_Check
                                                             :oneD:
  Rows_A, Rows_B, Columns_A, Columns_B
                                                             :integer;
  MinHam, Num_Patterns, Length_A, Length_B
                                                             :integer;
  Memory
                                                             :square;
  topline, bottomline, margin, leftone, rightone
                                                             :integer;
  lefttwo,leftthree,righttwo,rightthree,leftfour
                                                             :integer;
                                                             :real;
  threshold, pattern_number, TL, LL, AR, AC, PAC, PAR
                                                             :integer;
  test_type, Matrix_Used
                                                             :char:
  Synchmode, input, input_a, input_b
                                                             :boolean;
  inputfile, outfle
                                                             :text:
  filename, filename2
                                                             :string[10];
{$I xface.inc}
{ **********
function max(x,y: integer): integer;
begin
  if x > y then max := x
         else max := y;
end:
function min(x,y: integer): integer;
```

```
begin
 if x < y then min := x
       else min := y;
end:
procedure zero_test;
 index:integer;
begin
 for index := 1 to maxentries do
   TestPattern[index] := 0;
                                   { zero out Test matrix
end:
Procedure Readcr(var charValue, errorCode: integer);
{Read the screen at cursor position}
type
 RegPack = record
        AL, AH, BL, BH, CL, CH, DL, DH
                               : Byte:
        BP,SI,DI,DS,ES,Flags
                               : Integer;
        end:
var
 Regs
                : RegPack;
begin
with Regs do
          begin
            Errorcode:=0; {assume no error}
                         {code 8- screen read, page 0}
            AH:=$8: BH:=$0:
            Intr($10, Regs);
                          {get character in AL via int 10h}
                         {used to be AL - 48!!!!!}
            charValue: = AL;
end: {Reader}
procedure Read_Row_and_Column_Values;
var
x:integer;
begin
repeat
 textbackground(lightcyan);
                        { clears out any predefined user background }
 clrscr;
 textmode(C80);
 textbackground(lightcyan);
 textcolor(red);
 GoToXY(8,4);
 write('BIDIRECTIONAL ASSOCIATIVE MEMORY');
 GoToXY(8,7); Textcolor(blue);
 write ('Enter the number of patterns to store.
                                     <1..', maxpatterns, '>');
 readln(num_patterns);
 GoToXY(8,8);
 write ('Enter the number of rows in pattern A:
                                      <1...'.maxrows.'>');
 readln(rows_a);
 GoToXY(8,9);
 write('Enter the number of columns in pattern A: < 1..', maxcolumns, '>');
 readln(columns_a);
 GoToXY(8,10);
 write('Enter the number of rows in pattern B:
                                     <1..', maxrows, '>');
 readln(rows_b);
 GoToXY(8,11);
 write('Enter the number of columns in pattern B: < 1..', maxcolumns,'>');
 readln(columns_b);
 GoToXY(8,12);
 writeln('Enter the threshold of neuron activation:');
 GoToXY(10,13);
 write(' Value must be in range: - ', maxentries, ', + ', maxentries, ' ');
 readln(threshold);
 Length_A := Rows_A * Columns_A;
 Length_B := Rows_B * columns_B;
```

```
TextColor(Red + Blink); { blinks if inputs are unacceptable }
 if Length_A <= maxentries then input_a := True
   else
   begin
    input_a := False;
    GoToXY(13,17);
    writeln('Values for matrix A are out of bounds.');
    repeat until keypressed;
   if Length_B <= maxentries then input_b := True
    else
    begin
      input_b := False;
      GoToXY(13,17);
      writeln('Values for matrix B are out of bounds.');
      repeat until keypressed;
   if num_patterns < min(length_a,length_b) then input := True
    else
    begin
      input := False;
      GoToXY(13,17);
      writeln('Number of patterns must be less than ',min(length_a,length_b));
      repeat until keypressed;
   TextColor(Blue);
 until (input_a and input_b and input); { all inputs are within range }
end;
      procedure UseCurrentScreenSetup;
begin
   Synchmode := True;
          := 5;
   topline
  bottomline := 15;
  margin := 3;
  leftone
  rightone := 18;
  lefttwo := 22;
  righttwo := 36;
  leftthree := 40;
  rightthree := 54;
  leftfour := 58;
procedure SetMemoryToZero;
{$R+,V+,K+,C-,U-}
 index, row, column, size: integer;
begin
 size := max(length_a,length_b);
 for row := 1 to size do
   for column := 1 to size do
    memory[row,column] := 0;
Procedure SaveScreen(Matrix_Used:char; row_in, column_in:integer);
 position, charValue, ErrorCode
                                     :integer;
begin
 position := 1;
   for AR := 1 to row_in do
    begin
      for AC := 1 to column_in do
       begin
         PAC:=LL+AC-1;
```

```
PAR:=TL+AR-1;
            GoToXY(PAC, PAR);
            Readcr(charValue, ErrorCode); Delay(2);
            if ErrorCode <> 0 then write('error');
            case Matrix Used of
              'A': begin
                  if charValue = 177 then
                   Pattern_A[Pattern_Number, 1, position] := 1
                  else
                   Pattern_A[Pattern_Number, 1, position] := 0;
                 end;
              'B': begin
                  if charValue = 177 then
                   Pattern_B[Pattern_Number, 1, position] := 1
                  else
                   Pattern_B[Pattern_Number, 1, position] := 0;
                 end;
              'T': begin
                  if charValue = 177 then
                   TestPattern[position] := 1
                  else
                   TestPattern[position] := 0;
                 end;
            end; { end case }
            position := position + 1;
          end;
      end:
 { the following text erases the instructions yet leaves the Test Pattern }
 if Matrix_Used = 'T' then
 begin
   TextBackground(lightcyan);
   GoToXY(1,BottomLine + 4); { Beginning of instructions on screen }
   writeln('
                                     1);
   writeln('
                                        1);
                                       1);
   writeln('
                                           1);
   writeln('
 end;
Procedure DataFromKeyboard (Matrix_Used:char; rows,columns:integer);
{PatternNumber must be defined prior to call}
var
char3
                           :char;
intval
                          :integer;
charValue
                          :integer;
label
    loop1, InitLoop;
begin
 TextBackground(lightgray);
 GoToXY(1,1);
  { only print heading for the first time this screen appears i.e., Matrix_a }
  if Matrix_Used <> 'B' then
  case Pattern_Number of
               0: begin
                   write(' Enter the Test Pattern');
                  end;
               1.. MaxPatterns:
                  begin
                   TextColor(blue);
                   write(' Enter Pattern Number ',pattern_Number:2);
 end; {case}
 TextColor(blue);
 TextBackground(lightcyan);
```

```
case Matrix Used of
        'A' : begin
                                         { Matrix A input }
                GoToXY(LL,TL-2);
                write('MATRIX A');
             end:
        'B' : begin
                                         { Matrix B input }
                GoToXY(lefttwo,TL-2);
                write('MATRIX B');
             end:
        'T' : begin
                                         { TestPattern input }
                GoToXY(LL,TL-2);
                write('TEST PATTERN');
             end:
   end; { end case }
  TextColor(Magenta);
  TextBackground(lightgray);
       for AR:= 1 to Rows do
         begin
          for AC: = 1 to Columns do
           begin
             PAC:=LL+AC-1; {column to place cursor}
             PAR:=TL+AR-1; {row to place cursor}
             GoToXY(PAC, PAR);
             write(chr(249));
             GoToXY(PAC,PAR); { cursor stays in position }
           end;
       end;
{A zero matrix is now on the screen for Pattern 'PatternNumber'}
 TextColor(blue);
                                  {INSTRUCTIONS}
 TextBackground(lightcyan);
 GoToXY(1,BottomLine + 4);
                                  { Next free line on screen }
 writeln('
                           Position cursor using arrow keys.');
 writeln('
                           Press period "." to change pattern.');
 writeln('
                           Press space bar to remove changes. ');
 write(' Press RETURN to store Matrix after entering complete pattern');
 Textbackground(lightgray);
 InitLoop: GoToXY(LL,TL); { cursor to first element of input pattern}
               {initialize row and column counters}
 AC:=LL;
 AR:=TL;
loop1:read(kbd,char3);
      intval:=ord(char3);
      if intval = 27 then
                    begin
                      read(kbd,char3);
                      intval:=ord(char3);
   case intval of
                          { beeps on attempt to move off pattern display }
       80: begin
             if AR + 1 >= Rows + TL then
              begin sound(800); delay(60); nosound; end
              else AR := AR+1; { down arrow}
           end;
       72: begin
            if AR - 1 < TL then
             begin sound(800); delay(60); nosound; end
             else AR := AR-1; {up arrow}
           end:
       75: begin
            if AC - 1 < LL then
            begin sound(800); delay(60); nosound; end
            else AC := AC-1; {left arrow}
           end;
       77: begin
            if AC + 1 >= Columns + LL then
            begin sound(800); delay(60); nosound; end
            else AC := AC+1; {right arrow}
           end:
```

```
46: begin
                         {digits}
            write(chr(177));
           end;
       32: begin
            textcolor(magenta);
            write(chr(249));
            textcolor(blue);
           end;
       13: begin
             SaveScreen(Matrix_Used,Rows,Columns); { works for Matrix A, B or Test }
           end; {of case 13}
  end; {case statement}
  GoToXY(AC, AR); {goto new cursor position}
  if intval <> 13 then goto loop1;
TextBackground(lightcyan);
end; {DataFromKeyboard}
procedure EraseOldMatrices;
begin
  TextBackground(lightcyan);
{ clear old Matrix A }
 LL := leftone;
 for AR: = 1 to Rows_A do
 begin
   for AC: = 1 to Columns_A do
   begin
     PAC:=LL+AC-1; {column to place cursor}
     PAR:=TL+AR-1; {row to place cursor}
     GoToXY(PAC, PAR);
     write('');
   end;
 end;
{ clear old Matrix B }
 LL := lefttwo;
 for AR: = 1 to Rows_B do
 begin
   for AC: = 1 to Columns_B do
   begin
     PAC:=LL+AC-1; {column to place cursor}
     PAR:=TL+AR-1; {row to place cursor}
     GOTOXY(PAC, PAR);
     write('');
   end;
 end;
end;
Procedure InputTestPattern;
var
 n:integer;
begin
 Pattern_Number := 0;
   for n := 1 to maxentries do
   begin
     A_Check[n] := 0;
     B\_Check[n] := 0;
   end;
   TextColor(Red);
   repeat
      GoToXY(1,Bottomline + 2);
writeln('
                                                                                      1);
writeln(
writeln('
                                                                                      1);
                                                                                      1);
writeln('
```

```
GoToXY(1,Bottomline + 2);
          write('
                              Is test pattern of type A or B? (A/B) ');
          readln(test_type);
       until test_type in ['a','A','b','B'];
                                          { erases 'Is test pattern of ' query }
       GoToXY(1,Bottomline + 2);
       writeln('
       { eliminate lowercase input and relace with uppercase equivalent }
       if test_type = 'a' then test_type := 'A';
       if test_type = 'b' then test_type := 'B';
       if test_type = 'A' then
       begin
         clrscr;
         TL := topline; LL := leftone;
        DataFromKeyboard('T',rows_a,columns_a);
      else
      begin { if test_type = B }
        clrscr;
         TL := topline;
                            LL := leftone;
        DataFromKeyboard('T',rows_b,columns_b);
  OriginalTestPattern := TestPattern;
end;
{ *****
  **************
procedure ComputeEnergy;
var
  sum : real;
  temp: oneD;
  pattern_n,len_A,len_B,temp1: integer;
begin
 sum := 0.0;
 energy := 0.0;
 for pattern_n := 1 to num_patterns do
  begin
   for len_B := 1 to Length_B do
    begin
     temp[len_B] := 0;
     for len_A := 1 to Length_A do
temp1 := ( memory[len_A,len_B] - 0 );
      temp[len_B] := temp[len_B] + Pattern_A[pattern_n,1,len_A] * temp1;
      end;
    end;
   for len_B := 1 to Length_B do
      sum := sum + temp[len_B] * Pattern_B[pattern_n,1,len_B];
   energy := -sum;
  end;
end:
{ *********************
procedure Hamming;
var
 n,j:integer;
for n := 1 to num_patterns do
   Ham[n] := 0;
MinHam := 1;
for n := 1 to num_patterns do
 begin
   if test_type = 'A' then
   begin
   for j := 1 to Length_A do
     if Pattern_A[n,1,j] <> OriginalTestPattern[j]
        then Ham[n] := Ham[n] + 1;
     if Ham[n] < Ham[MinHam] then MinHam := n;
```

```
end
   else
   begin
     for j := 1 to Length_B do
       if Pattern_B[n,1,j] <> OriginalTestPattern[j]
        then Ham[n] := Ham[n] + 1;
     if Ham[n] < Ham[MinHam] then MinHam := n;
   end:
end:
{ ********
                           *****************
procedure status(x,y:integer;TxT:textin);
var first: char;
   last:textin;
begin
 TextBackground(blue); { if status is not called from StatusLine }
  first := copy(TxT,1,1);
 last := copy(TxT,2,(length(TxT) - 1));
                                       write(first);
 GoToXY(x,y);
                  Textcolor(white);
 GoToXY(x+1,y);
                  Textcolor(yellow);
                                       write(last);
end:
Procedure StatusLine;
vari:integer; ch:char;
begin
TextBackground(blue);
GoToXY(1,23);
 for i := 1 to screencolumns do write(''); { status line background }
GoToXY(12,23); TextColor(Yellow);
write('STATUS LINE - First letter of choice and RETURN selects:');
GoToXY(1,24);
 for i := 1 to screencolumns do write(''); { status line background }
Status(15,24,'Quit');
if SynchMode = True then Status(34,24, 'Synch')
                else Status(34,24,'Asynch');
Status(55,24, 'Ham dist');
GoToXY(15,20);
write('Select execution Mode -- Synchronous/Asynchronous');
repeat
  begin
    read(kbd,ch);
    if ch in ['s', 'S'] then SynchMode := True;
    if ch in ['a', 'A'] then SynchMode := False;
 until ch in ['a', 'A', 's', 'S'];
 if SynchMode = True then Status(34,24, 'Synch')
                else Status(34,24,'Asynch');
Textbackground(lightcyan); Textcolor(blue);
GoToXY(15,20);
write('
                                                       1);
end; {StatusLine}
Procedure TurnPCcursorOff;
{get rid of regular cursor}
type
   RegPack = record
           AL, AH, BL, BH, CL, CH, DL, DH: Byte;
           BP,SI,DI,DS,ES,Flags : Integer;
           end;
var
                   : RegPack;
 Regs
```

```
begin
    with Regs do
                begin
                 AH:=$1; CH:=16; CL:= 0;
                 Intr($10, Regs);
                end;
  end; {TurnPCcursorOff}
  { *****************************
   *************
 Procedure TurnPCcursorOn;
  {turn on regular cursor}
  type
      RegPack = record
               AL, AH, BL, BH, CL, CH, DL, DH: Byte;
               BP,SI,DI,DS,ES,Flags : Integer;
 var
   Regs
                         : RegPack;
  begin
   with Regs do
                begin
                 AH:=$1; CH:=7; CL:= 9; {start line>end means cursor off}
                 Intr($10,Regs);
                end;
   end; {TurnPCcursorOn}
 procedure BipolarizeB;
   index:integer;
begin
   for index := 1 to Length_B do
     begin
       if Pattern_B[Pattern_Number, 1, index] = 0
         then Bipolar_B[Pattern_Number, 1, index] := -1
         else Bipolar_B[Pattern_Number, 1, index] := 1;
     end:
end;
procedure BipolarizeA;
  index:integer;
begin
  for index := 1 to Length_A do
    begin
      if Pattern_A[Pattern_Number, 1, index] = 0
        then Bipolar_A[Pattern_Number, 1, index] := -1
        else Bipolar_A[Pattern_Number, 1, index] := 1;
end;
{ ************
procedure Memorize_Bipolar;
var
  pattern_n,len_A,len_B,temp: integer;
begin
GoToXY(4,bottomline + 2);
 write(' * Please wait - Bipolarization in Progress * ');
 for pattern_n := 1 to num_patterns do
 for len_A := 1 to Length_A do
   for len_B := 1 to Length_B do
     memory[len_a,len_b] := memory[len_a,len_b] +
                            bipolar_a[pattern_n,1,len_a] *
                            bipolar_b[pattern_n,1,len_b];
```

```
end;
function CheckIfKeypressed: boolean;
var
 ch:char; n, keyint: integer;
begin
  if not(keypressed) then CheckIfKeypressed := False
    begin
      read(kbd,ch);
      keyint := ord(ch);
      case keyint of
          113,81 : CheckIfKeypressed := True;
                  { Q or q for Quit has been pressed }
          115,83: begin {Sors}
                    Synchmode := True;
                    Status(34,24, 'Synch');
                    CheckIfKeypressed := False; {continues execution}
           97,65: begin { A or a }
                    Synchmode := False;
                    Status(34,24,'Asynch');
                    CheckIfKeypressed := False; {continues execution}
          104,72 : begin
                      for n := 1 to num_patterns do
                        begin
                          GoToXY(16,18+(n-1));
                          write('Hamming Distance for Pattern ' ,test_type);
                          writeln('',n,' is:',Ham[n]);
                        end;
                        GoToXY(20,22);
                        writeln('Press any key to continue');
                      repeat until keypressed;
                        GoToXY(20,22);
                                                        1);
                      CheckIfKeypressed := False; {continues execution}
                      for n := 1 to num_patterns do
                        begin
                          GoToXY(16,18 + (n-1));
                                                               1);
                          write('
                          writeln('
                                                     1);
                        end:
          else CheckIfKeypressed := False;
                  { no action taken }
      end; { end case }
    end;
end;
Procedure Bammer(Test_Pat:oneD; test_now:char;
                               leng1, leng2: integer);
var
  Memory_Transpose
                                    :Square;
  maxrow,n,k,i,j,m,y,Start,Finish
                                   :integer;
                                    :OneD;
  BinVect
begin
 Textbackground(lightcyan);
 TextColor(blue);
 GoToXY(lefttwo,TL-2);
 write('MATRIX A');
 GoToXY(leftthree, TL-2);
 write('MATRIX B');
```

```
Textbackground(lightgray);
TextColor(magenta);
if Synchmode = True then
  begin
                     { Synchronous Mode prints out all Neurons }
    for j := 1 to leng1 do
         OutPatt[j] := 0;
    case test_now of
    'A' : begin
           B_Check := Test_Pat;
            for i := 1 to leng2 do
           if Test_Pat[i] = 0 then BinVect[i] := -1
                              else BinVect[i] := 1;
           for i := 1 to leng1 do
              for j := 1 to leng2 do
                Memory_Transpose[j,i] := Memory[i,j];
           for j := 1 to leng1 do
             for i := 1 to leng2 do
                OutPatt[j] := OutPatt[j] + BinVect[i] * Memory_Transpose[i,j];
           end:
    'B' : begin
           A_Check := Test_Pat;
           for i := 1 to leng2 do
            if Test_Pat[i] = 0 then BinVect[i] := -1
                               else BinVect[i] := 1;
           for i := 1 to leng1 do
             for j := 1 to leng2 do
               OutPatt[i] := OutPatt[i] + BinVect[j] * Memory[j,i];
           end;
end; { end case }
for j := 1 to leng1 do
begin
  if (OutPatt[j] - threshold) > 0 then TestPattern[j] := 1
  if (OutPatt[j] - threshold) < 0 then TestPattern[j] := 0</pre>
  else
    case test_now of
      'A' : TestPattern[j] := A_Check[j];
       'B' : TestPattern[j] := B_Check[j];
    end; { end case }
end; { end for Start to Finish }
case test_now of
'A':
    begin
      k := TL - 1;
                                             { prints out matrix A }
      for y := 0 to Length_A - 1 do
      begin
        i := y mod Columns_A;
        if i = 0 then k := k + 1;
        GoToXY(lefttwo + i,k);
        if TestPattern[y + 1] = 1 then write(chr(177))
                                  else write(chr(249));
      end;
     k := TL - 1;
                                            { prints out matrix B }
      for y := 0 to Length_B - 1 do
       i := y mod Columns_B;
       if i = 0 then k := k + 1;
       GoToXY(leftthree + i,k);
       if Test_Pat[y+1] = 1 then write(chr(177))
                             else write(chr(249));
     end;
    end; { 'A' }
  'B':
   begin
```

```
k := 1;
                                       { prints out matrix A }
     for i := 0 to Rows_A - 1 do
     begin
       for j := 0 to Columns_A - 1 do
       begin
         GoToXY(lefttwo+j,TL+i);
         if Test_Pat[k] = 1 then write(chr(177))
                            else write(chr(249));
         k := k + 1;
       end;
     end;
     k := 1;
                                       { prints out matrix B }
     for i := 0 to Rows_B - 1 do
       for j := 0 to Columns_B - 1 do
         GoToXY(leftthree + j,TL + i);
          if TestPattern[k] = 1 then write(chr(177))
                              else write(chr(249));
         k := k + 1;
       end;
     end;
   end; { 'B' }
  end; { case test_now of }
end; { if Synchmode true }
if Synchmode = False then
                   { Asynchronous Mode prints out one Neuron }
  for j := 1 to leng1 do
      OutPatt[j] := 0;
Case test_now of
  'A' : begin
         B_Check := Test_Pat;
         for i := 1 to leng2 do
         if Test_Pat[i] = 0 then BinVect[i] := -1
                            else BinVect[i] := 1;
         for i := 1 to leng1 do
           for j := 1 to leng2 do
             Memory_Transpose[j,i] := Memory[i,j];
         Start := Random(leng1);
         if Start = 0 then Start := leng1;
         for i := 1 to leng2 do
           OutPatt[Start] := OutPatt[Start] +
                              BinVect[i] * Memory_Transpose[i,Start];
       end;
  'B' : begin
         A_Check := Test_Pat;
         for i := 1 to leng2 do
         if Test_Pat[i] = 0 then BinVect[i] := -1
                            else BinVect[i] := 1;
         Start := Random(leng1);
         if Start = 0 then Start := leng1;
         for i := 1 to leng2 do
         OutPatt[Start] := OutPatt[Start] +
                            BinVect[i] * Memory[i,Start];
       end;
 end; { end case }
  for j := 1 to leng1 do
   if (OutPatt[j] - threshold) > 0 then TestPattern[j] := 1
    else
   if (OutPatt[j] - threshold) < 0 then TestPattern[j] := 0</pre>
   else
```

```
case test_now of
               'A' : TestPattern[j] := A_Check[j];
               'B' : TestPattern[j] := B_Check[j];
           end; { end case }
      end; { end for Start to Finish }
      case test_now of
       'A' : begin
              k := TL - 1:
                                                  { prints out matrix A }
              for y := 0 to Length_A - 1 do
              begin
                i := y mod Columns_A;
                if i = 0 then k := k + 1;
                GoToXY(lefttwo + i,k);
                if TestPattern[y + 1] = 1 then write(chr(177))
                                       else write(chr(249));
            end;
            k := TL - 1; { prints out matrix B }
           for y := 0 to Length_B - 1 do
           begin
             i := y mod Columns_B;
             if i = 0 then k := k + 1;
             GoToXY(leftthree + i,k);
             if Test_Pat[y + 1] = 1 then write(chr(177))
                                  else write(chr(249));
           end;
      end; { 'A' }
      'B' : begin
           k := 1;
                                           { prints out matrix A }
           for i := 0 to Rows_A - 1 do
           begin
             for j := 0 to Columns_A - 1 do
             begin
               GoToXY(lefttwo+j,TL+i);
               if Test_Pat[k] = 1 then write(chr(177))
                                else write(chr(249));
              k := k + 1;
             end;
           end;
           k := 1;
                                           { prints out matrix B }
           for i := 0 to Rows_B - 1 do
           begin
             for j := 0 to Columns_B - 1 do
               begin
                 GoToXY(leftthree + j, TL + i);
                 if TestPattern[k] = 1 then write(chr(177))
                                     else write(chr(249));
                 k := k + 1;
              end;
          end;
          end; { 'B' }
     end; { case test_now of }
   end; { end if Synchmode False }
 GoToXY(1,Bottomline + 2);
 TextBackground(lightcyan);
 TextColor(blue);
                     procedure Bam;
begin
GoToXY(1,1);
                      TextColor(Red);
write('
                                PROCESSING');
if test_type = 'A' then
  begin
    repeat
```

```
Bammer(TestPattern, 'B', Length_B, Length_A);
      Bammer(TestPattern, 'A', Length_A, Length_B);
     until CheckIfKeypressed;
  end
  else
  begin
     repeat
       Bammer(TestPattern, 'A', Length_A, Length_B);
      Bammer(TestPattern, 'B', Length_B, Length_A);
     until CheckIfKeypressed;
  end;
end;
function DataFromFile: boolean;
begin
  textbackground(lightcyan);
  clrscr;
                     { clears out any predefined user background }
  textmode(C80);
  textbackground(lightcyan);
  textcolor(red);
  GoToXY(8,4);
  write('BIDIRECTIONAL ASSOCIATIVE MEMORY');
   DataFromFile := False;
  GoToXY(1,8);
  if yes('
                  Do you want to read the patterns from a file? ') then
  begin
   GoToXY(1,9);
   write('
                   Enter the filename to read from: ');
   readln(filename);
   assign(inputfile, filename);
    {$I-}
   reset(inputfile);
    {\$I+}
   DataFromFile := True;
   if not(ioresult = 0) then
   begin
     GoToXY(1,9); Textcolor(Red + Blink);
     writeln('Unable to open file ');
     exit;
   end;
  end;
procedure ReadInFile;
var
  temp:integer; ch:char;
begin
  TurnPCcursorOff;
  readln(inputfile,num_patterns);
  readln(inputfile, Rows_A);
  readln(inputfile,Columns_A);
  readln(inputfile, Rows_B);
  readln(inputfile,Columns_B);
  Length_A := Rows_A * Columns_A;
  Length_B := Rows_B * columns_B;
  clrscr;
  textcolor(blue);
                          TL := topline;
  GoToXY(1,1);
  write('Reading Patterns from file: ',filename);
  GoToXY(leftone, TL - 2);
  write('MATRIX A');
  GoToXY(lefttwo,TL-2);
  write('MATRIXB');
  textbackground(lightgray);
  textcolor(magenta);
  for Pattern_Number := 1 to num_patterns do
  begin
  GoToXY(13,bottomline + 1);
  write('Pattern',Pattern_Number);
```

```
TL := topline;
                               LL := leftone:
       for AR := 1 to Rows_A do
         begin
         for AC := 1 to Columns_A do
           begin
             PAC := LL + AC - 1;
             PAR := TL + AR - 1;
             GOTOXY(PAC, PAR);
             read(inputfile, temp);
             if temp = 1 then write(chr(177))
                        else write(chr(249));
           end;
         end:
      LL := lefttwo; TL := topline;
      for AR := 1 to Rows_B do
        begin
        for AC := 1 to Columns_B do
          begin
            PAC := LL + AC - 1;
            PAR := TL + AR - 1;
            GOTOXY(PAC, PAR);
            read(inputfile, temp);
            if temp = 1 then write(chr(177))
                      else write(chr(249));
         end;
       end;
      LL := leftone; TL := topline;
      SaveScreen('A', Rows_A, Columns_A);
      LL := lefttwo; TL := topline;
      SaveScreen('B', Rows_B, Columns_B);
      BipolarizeB:
      BipolarizeA;
      GoToXY(8,bottomline + 2);
                                 Textbackground(lightcyan);
      Textcolor(red); write('Press any key to continue reading patterns.');
      repeat until keypressed;
      read(kbd,ch);
      GoToXY(8, bottomline + 2);
                                   Textbackground(lightcyan);
      write('
                                       '); { erase above }
      Textcolor(magenta); Textbackground(lightgray);
    end; { for Pattern_Number }
    textcolor(blue);
                                 Textbackground(lightcyan);
   GoToXY(4, bottomline + 1);
                                    TurnPCcursorOn;
   writeln('Enter the threshold of neuron activation:');
   GoToXY(4,bottomline + 2);
   write(' Value must be in range: - ', maxentries, ', + ', maxentries, ' ');
   readln(threshold);
                                   TurnPCcursorOff;
   GoToXY(4, bottomline + 1);
   writeln('
                                       1);
Procedure WriteToFile:
varn, z: integer;
begin
  if yes('Do you want to save the memory patterns to a file?') then
     begin
       write('Enter the filename to save patterns to: ');
       readln(filename2);
       assign(outfle,filename2);
       rewrite(outfle);
        writeln(outfle,num_patterns);
        writeln(outfle,Rows_A);
        writeln(outfle,Columns_A);
        writeln(outfle, Rows_B);
        writeln(outfle,Columns_B);
        for n := 1 to num_patterns do
         begin
            for z := 1 to Length_A do
              write(outfle,Pattern_A[n,1,z],'');
            writeln(outfle);
```

```
for z := 1 to Length_B do
            write(outfle,Pattern_B[n,1,z],'');
          writeln(outfle);
         end;
       close(outfle);
   end;
end:
begin { MAIN }
repeat { until not 'yes try another set of patterns ' }
 UseCurrentScreenSetup;
 if DataFromFile = False then
 begin { input is from the keyboard }
   Read_Row_and_Column_Values;
   SetMemoryToZero;
   clrscr;
   for Pattern_Number := 1 to num_patterns do
    begin
     TL := topline;
                      LL := leftone;
     DataFromKeyboard('A',rows_a,columns_a);
                      LL := lefttwo;
     TL := topline;
     DataFromKeyboard('B',rows_b,columns_b);
     BipolarizeB;
     BipolarizeA;
     if Pattern_Number <> num_patterns then EraseOldMatrices;
   end:
  end { input is from the keyboard
  else { input is from the files
   ReadInFile;
   SetMemoryToZero;
  Memorize_Bipolar;
  ComputeEnergy;
  TurnPCcursorOn;
  repeat { until not yes 'another test pattern ' }
   zero_test;
   InputTestPattern;
   Hamming;
   TurnPCcursorOff;
   StatusLine;
   Bam;
   TurnPCcursorOn;
   WriteToFile;
 until not yes(' Do you want to try another test pattern?');
until not yes(' Do you want to try another set of patterns?');
TextMode; { returns screen to previous graphics color mode }
clrscr;
end.
```

XFACE.INC From "Constructing an Associative Memory" by Bart Kosco, BYTE, September 1987.

```
type
   mstring = string[100];
var
   outfile : text;

FUNCTION yes(PROMPT:MSTRING): BOOLEAN;
var
      ch : string[2];
begin
      (*$I-*)(*$R-*)
      repeat
      write(prompt,'(y/n)');
      readln(ch);
      until (ch = 'y') or (ch = 'Y') or (ch = 'N') or (ch = 'n') and (ioresult=0);
      (*$I+*)(*$R+*)
      yes := (ch = 'y') or (ch = 'Y');
end;
```

```
PROCEDURE setoutfile;
    var
      ch: char;
    begin
      (*$I-*)(*$R-*)
      repeat
       write('OUTPUT DESTINATION: P(rinter C(onsole: ');
       readln(ch);
      until (ch in['c','C','p','P']) and (ioresult = 0);
      case ch of
        'p','P': ASSIGN(outfile,'LST:');
'c','C': ASSIGN(outfile,'CON:');
      end; (*case*)
      REWRITE(OUTFILE);
      (*$I+*)(*$R+*)
    end; (*setoutfile*)
   FUNCTION getnum(PROMPT: MSTRING; LOW, HIGH: INTEGER): INTEGER;
    var
      val: integer;
   begin
      VAL := -31695;
      (*$I-*)(*$R-*)
      repeat
        write(prompt, '(', low, '..', high, '): ');
        readln(val);
        if (val < low) or (val > high)
         then writeln(' VALUE OUT OF RANGE');
        if ioresult <> 0
          then writeln('WRONG DATA TYPE');
     until (val >= low) and (val <= high) and (ioresult = 0);
     getnum := val;
      (*$I+*)(*$R+*)
   end:
 FUNCTION getchar(PROMPT: MSTRING): CHAR;
  ch : char;
  begin
    (*$I-*)(*$R-*)
    repeat
      write(prompt);
      readln(ch);
    until (ch IN ['a'..'z','A'..'Z']) and (ioresult = 0);
    getchar := ch;
    (*$I+*)(*$R+*)
  end;
FUNCTION getreal(PROMPT: MSTRING; LOW, HIGH: REAL): REAL;
 var
   val: real;
 begin
   VAL := -31695.7;
   (*$I-*)(*$R-*)
   repeat
     write(prompt, '(',low:3:1,'..',high:3:1,'):');
      readln(val);
      if (val < low) or (val > high) or (ioresult <> 0)
        then writeln('Data incorrect type or out of range');
   until (val <= low) and (val <= high) and (ioresult = 0);
      getreal := val;
      (*$I+*)(*$R+*)
 end;
```

BAM.BAS From "Constructing an Associative Memory" by Bart Kosco, BYTE, September 1987.

```
1000 CLS
1010 PRINT
1020 PRINT
1030 PRINT
1040 PRINT "
                       1050 PRINT "
                                                                                          *"
1060 PRINT "
                                                                                          *"
                                        BIDIRECTIONAL ASSOCIATIVE MEMORY
1070 PRINT "
                                                                                          *"
                                              DEMONSTRAION PROGRAM
1080 PRINT "
                                                                                          * "
1090 PRINT "
                                (C) COPYRIGHT 1987 LOGICAL DESIGNS CONSULTING INC.
                                                                                          * "
1100 PRINT "
                                                                                          * "
                                         3229 ERIE ST. SAN DIEGO, CA 92117
1110 PRINT "
                                                                                          *"
                                                  (619) 276-3955
1120 PRINT "
                                                                                          * II
1130 PRINT "
                                                                                          * "
                                                 BY DUANE DESIENO
1140 PRINT "
                                                                                          *"
1150 PRINT "
                      **********
1160 PRINT
1170 PRINT
1180 PRINT
1190 PRINT
                      PRESS (Y) FOR INSTRUCTIONS"
1200 PRINT "
1210 S$=INKEY$
1220 IF LEN(S$) = 0 THEN GOTO 1210
1230 IF S$<>"Y" AND S$<>"y" THEN GOTO 1760
1240 CLS
1250 PRINT
1260 PRINT "
                      USING THE BAM DEMONSTRATION PROGRAM"
1270 PRINT
1280 PRINT " - CHANGE NETWORK PARAMETERS to set new values of the A and B"
1290 PRINT " dimensions, the number of cells updated per iteration of the"
1300 PRINT " network, and the percentage of elements to change state when"
1310 PRINT " random noise is added to the A and B fields. The maximun"
1320 PRINT " size of the A or B fields is 144 elements (12x12)."
1330 PRINT
1340 PRINT " - CLEAR NETWORK fills the matrix M with O. All stored patterns"
1350 PRINT " will be lost when this command is executed, so be sure to"
1360 PRINT " save the M matrix before executing this command."
1370 PRINT
1380 PRINT " - LOAD MEMORY MATRIX M displays all current BAM interconnect"
1390 PRINT " matrix files stored on disk. Enter the desired filename"
1400 PRINT " to load that file into the M matrix."
1410 PRINT
1420 PRINT " - SAVE MEMORY MATRIX M to store the current M matrix to a disk"
1430 PRINT " file. The dimensions of the A and B fields are also saved."
1440 PRINT
1450 PRINT
1460 PRINT "
                      PRESS ANY KEY TO CONTINUE"
1470 S$=INKEY$
1480 IF LEN(S$)=0 THEN GOTO 1470
1490 CLS
1500 PRINT
1510 PRINT " - EDIT/RUN NETWORK to input new patterns to the A and B fields."
1520 PRINT " Once input, the network can either learn the new pattern or"
1530 PRINT " execute one or more iterations of the network.
1540 PRINT
1550 PRINT " - LEARN CURRENT PATTERN takes the current state of the A and B"
1560 PRINT " fields and changes the M matrix to learn this pattern. The"
1570 PRINT " cursor will disappear until this operation is complete.
1580 PRINT
1590 PRINT " - ADD RANDOM NOISE will flip the state of a certain percentage"
1600 PRINT " of elements in both the A and B fields. The percentage is"
1610 PRINT " set in the NETWORK PARAMETERS. This can be used to see how"
1620 PRINT " different a pattern can be to still be recalled.
1630 PRINT
1640 PRINT " - RUN THE NETWORK will execute two complete iterations of the"
1650 PRINT " network when the parameter is set for synchronous operation."
1660 PRINT " When the number of cells updated per iteration is greater "
             than 0, 10 iterations of the network are executed. At each"
1680 PRINT " iteration, only the number of cells specified in the parameter"
1690 PRINT " are updated per field."
1700 PRINT
1710 PRINT
```

```
1720 PRINT "
                       PRESS ANY KEY TO CONTINUE"
  1730 S$=INKEY$
  1740 IF LEN(S$)=0 THEN GOTO 1730
  1760 ON ERROR GOTO 4720
  1770 DIM A%(144), B%(144), X%(144), Y%(144), M%(144, 144)
 1780 AXSIZE=12: AYSIZE=12: BXSIZE=12: BYSIZE=12
 1790 CLS
 1800 KEY OFF
 1810 PRINT "
                                         BIDIRECTIONAL ASSOCIATIVE MEMORY "
 1820 PRINT "
                                              DEMONSTRATION PROGRAM"
 1830 PRINT
 1840 PRINT
 1850 PRINT "
                                                 BY DUANE DESTENO"
 1860 PRINT
 1870 PRINT
 1880 PRINT
 1890 PRINT "
                                                    MAIN MENU"
 1900 PRINT
 1910 PRINT "
                                          1 - CHANGE NETWORK PARAMETERS"
 1920 PRINT "
                                         2 - CLEAR NETWORK"
 1930 PRINT "
                                         3 - LOAD MEMORY MATRIX M"
 1940 PRINT "
                                          4 - SAVE MEMORY MATRIX M"
 1950 PRINT "
                                          5 - EDIT/RUN NETWORK"
 1960 PRINT "
                                         6 - QUIT"
 1970 PRINT
 1980 INPUT "
                                         INPUT CHOICE (1-6): ": CHOICE
         ON CHOICE GOSUB 2010,2370,4240,4480,3000,4750
 1990
 2000
        GOTO 1790 'THIS IS THE MAIN LOOP OF THE PROGRAM
 2010 REM ***************************
 2020 REM
                        CHANGE NETWORK PARAMETERS
 2040 CLS
 2050 LOCATE 5,1
 2060 PRINT "
                                         CURRENT NETWORK PARAMETERS"
 2070 PRINT
 2080 PRINT
 2090 PRINT "
                                   1 - A FIELD X DIMENSION
                                                                                 : "; AXSIZE
 2100 PRINT "
                                   2 - A FIELD Y DIMENSION
                                                                                 : "; AYSIZE
 2110 PRINT "
                                   3 - B FIELD X DIMENSION
                                                                                 : ";BXSIZE
 2120 PRINT "
                                   4 - B FIELD Y DIMENSION
                                                                                 : "; BYSIZE
 2130 PRINT "
                                   5 - NUMBER OF CELLS CHANGED PER "
2140 PRINT "
                                      ITERATION (0=SYNCHRONOUS)
                                                                                 : "; ASYN
2150 PRINT "
                                   6 - RANDOM NOISE PERCENTAGE
                                                                                 : "; NOISE
2160 PRINT "
                                   7 - RETURN"
2170 PRINT
2180 INPUT "
                      ENTER CHOICE :"; CHOICE
2190 IF CHOICE=7 THEN GOTO 2290
2200 INPUT "
                      ENTER NEW VALUE : "; NVAL
2210 ON CHOICE GOTO 2230,2240,2250,2260,2270,2280
2220 RETURN
2230 AXSIZE=NVAL: GOTO 2010
                                         ' LOOP TILL DONE
2240
       AYSIZE=NVAL: GOTO 2010
2250
       BXSIZE=NVAL: GOTO 2010
2260
       BYSIZE=NVAL: GOTO 2010
2270
      ASYN=NVAL: GOTO 2010
2280
       NOISE=NVAL: GOTO 2010
       IF AXSIZE*AYSIZE>144 THEN PRINT "A FIELD TOO LARGE";: GOTO 2050
2290
2300
      IF AYSIZE>16 THEN PRINT "A FIELD Y DIM TOO LARGE";: GOTO 2050
       IF BXSIZE*BYSIZE>144 THEN PRINT "B FIELD TOO LARGE";: GOTO 2050
2320 IF BYSIZE>16 THEN PRINT "B FIELD Y DIM TOO LARGE";: GOTO 2050
2330 RETURN
                    ' END OF CHANGE NETWORK PARAMETERS
2340 REM *********************************
2350 REM
                       CLEAR NETWORK MATRIX M, A, B
2360 REM *******************************
2370 FOR I=1 TO 144
2380 A%(I)=0
2390 B%(I)=0
2400 NEXT I
2410 FOR I=1 TO AXSIZE*AYSIZE 'CLEAR THE MEMORY MATRIX M
2420
     FOR J=1 TO BXSIZE*BYSIZE
2430
        M\%(I,J)=0
2440
      NEXT J
2450
      PRINT ".";
```

```
2460 NEXT I
2470 RETURN
                     ' END OF CLEAR NETWORK
2480 REM **********************
2490 REM
                DISPLAY A AND B FIELDS
2500 REM *********************
2510 CLS
2520 LOCATE 1,19: PRINT "A FIELD";
2530 LOCATE 1,59: PRINT "B FIELD";
2540 REM **** DRAW BOX AROUND A FIELD ****
2550 LOCATE 3,15: PRINT CHR$(218)
2560 FOR I=1 TO AXSIZE: LOCATE 3,15+I: PRINT CHR$(196);: NEXT I
2570 PRINT CHR$(191);
2580 FOR J=1 TO AYSIZE
2590 LOCATE 3+J,15: PRINT CHR$(179);
2600 LOCATE 3+J, 15+AXSIZE+1: PRINT CHR$(179);
2610 NEXT J
2620 LOCATE 3+AYSIZE+1,15: PRINT CHR$(192);
2630 FOR I=1 TO AXSIZE: LOCATE 3+AYSIZE+1,15+I: PRINT CHR$(196);: NEXT I
2640 PRINT CHR$(217);
2650 REM **** DRAW BOX AROUND B FIELD ****
2660 LOCATE 3,55: PRINT CHR$(218)
2670 FOR I=1 TO BXSIZE: LOCATE 3,55+I: PRINT CHR$(196);: NEXT I
2680 PRINT CHR$(191);
2690 FOR J=1 TO BYSIZE
2700 LOCATE 3+J,55: PRINT CHR$(179);
2710 LOCATE 3+J,55+BXSIZE+1: PRINT CHR$(179);
2720 NEXT J
2730 LOCATE 3+BYSIZE+1,55: PRINT CHR$(192)
2740 FOR I=1 TO BXSIZE: LOCATE 3+BYSIZE+1,55+I: PRINT CHR$(196);: NEXT I
2750 PRINT CHR$(217);
2760 LOCATE 21,1: PRINT "OPTIONS: 1. LEARN CURRENT PATTERN";
2770 LOCATE 21,41: PRINT "
                                   ARROW KEYS TO MOVE CURSOR";
2780 LOCATE 22,1:PRINT "
                                 ADD RANDOM NOISE";
2. ADD RANDOM NOISE";
2790 LOCATE 22,41:PRINT "
2800 LOCATE 23,1:PRINT "
2810 LOCATE 23,41:PRINT "
3. RUN THE NETWORK";
4. DETAIN "
4. TO SET LEVEL TO 1
                                   A OR B TO SWITCH FIELDS"
                                    + TO SET LEVEL TO 1";
2820 LOCATE 24,1:PRINT " 4. CLEAR A AND B FIELDS";
2830 LOCATE 24,41:PRINT " - TO SET LEVEL TO O";
2840 LOCATE 25,1:PRINT " ESC. RETURN TO MAIN MENU";
2850 REM **** DISPLAY THE A FIELD ARRAY IN THE BOX ****
2860 FOR J=1 TO AYSIZE
2870 FOR I=1 TO AXSIZE
2880
         LOCATE 3+J,15+I
         PRINT CHR$(219*A%((J-1)*AXSIZE+I));
2890
2900 NEXT I
2910 NEXT J
2920 REM **** DISPLAY THE B FIELD ARRAY IN THE BOX ****
2930 FOR J=1 TO BYSIZE
2940 FOR I=1 TO BXSIZE
2950
         LOCATE 3+J,55+I
2960
         PRINT CHR$(219*B%((J-1)*BXSIZE+I));
2970 NEXT I
2980 NEXT J
2990 RETURN
3000 REM ********************************
                        EDIT THE A AND B FIELDS
3010 REM
3020 REM **********************************
'DISPLAY THE FIELDS BEFORE EDITING
                           'START ON A FIELD UPPER LEFT CORNER
                                        'POSITION CURSOR IN FIELD BOX
3060 LOCATE 3+PY, 15+PX+40*FLD
3070 PRINT "*";
3080 FOR ZZ=1 TO 4: NEXT ZZ
3090 LOCATE 3+PY, 15+PX+40*FLD
3100 IF FLD=0 THEN XSIZE=AXSIZE: YSIZE=AYSIZE
3110 IF FLD=1 THEN XSIZE=BXSIZE: YSIZE=BYSIZE
3120 \text{ OFS}=((PY-1)*XSIZE+PX)
3130 IF FLD=0 THEN STAT=A%(OFS) ELSE STAT=B%(OFS)
3140 PRINT CHR$(219*STAT);
3150 IF LEN(S$)=2 THEN S$=RIGHT$(S$,1)
3160 IF S$=CHR$(77) THEN PX=PX+1
3170 IF S$=CHR$(75) THEN PX=PX-1
3180 IF S$=CHR$(72) THEN PY=PY-1
3190 IF S$=CHR$(80) THEN PY=PY+1
3200 IF S$="A" OR S$="a" THEN FLD=0: PX=1: PY=1
3210 IF S$="B" OR S$="b" THEN FLD=1: PX=1: PY=1
3220 IF S$="+" AND FLD=0 THEN A%(OFS)=1
```

```
3230 IF S$="+" AND FLD=1 THEN B%(OFS)=1
   3240 IF S$="-" AND FLD=0 THEN A%(OFS)=0
   3250 IF S$="-" AND FLD=1 THEN B%(OFS)=0
   3260 IF S$="1" THEN GOSUB 3370
                                           ' LEARN CURRENT PATTERN
   3270 IF S$="2" THEN GOSUB 3530
                                           ' ADD RANDOM NOISE TO PATTERN
   3280 IF S$="3" THEN GOSUB 3660
                                          ' RUN THE NETWORK
   3290 IF S$="4" THEN GOSUB 4150
                                          ' CLEAR THE A AND B FIELDS
   3300 IF PX<1 THEN PX=1
   3310 IF PX>XSIZE THEN PX=1: PY=PY+1
   3320 IF PY<1 THEN PY=1
   3330 IF PY>YSIZE THEN PY=YSIZE
  3340 IF S$=CHR$(27) THEN RETURN
  3350 FOR ZZ=1 TO 4: NEXT ZZ
  3360 GOTO 3050
  3370 REM **********************************
  3380 REM
                LEARN CURRENT PATTERN IN A AND B FIELDS
  3390 REM **********************************
  3400 FOR I=1 TO AXSIZE*AYSIZE TRANSFER A FIELD TO BIPOLAR X FIELD
  3410 IF A%(I)=0 THEN X%(I)=-1 ELSE X%(I)=1
  3420 NEXT I
  3430 FOR I=1 TO BXSIZE*BYSIZE
                                      ' TRANSFER B FIELD TO BIPOLAR Y FIELD
  3440 IF B%(I)=0 THEN Y%(I)=-1 ELSE Y%(I)=1
  3450 NEXT I
  3460 REM **** THE CORRELATION MATRIX M IS UPDATED HERE ****
  3470 FOR J=1 TO BXSIZE*BYSIZE
  3480 FOR I=1 TO AXSIZE*AYSIZE
  3490
         M\%(I,J)=M\%(I,J)+X\%(I)*Y\%(J)
  3500 NEXT I
  3510 NEXT J
  3520 RETURN
  3530 REM **********************************
             ADD RANDOM NOISE TO THE A AND B FIELDS
 3540 REM
 3550 REM **********************************
 3560 FOR I=1 TO AXSIZE*AYSIZE
 3570 IF 100*RND>=NOISE THEN 3590
      IF A%(I)=0 THEN A%(I)=1 ELSE A%(I)=0 'FLIP THE STATE
 3580
 3590 NEXT I
 3600 FOR I=1 TO BXSIZE*BYSIZE
 3610 IF 100*RND>=NOISE THEN 3630
 3620
       IF B%(I)=0 THEN B%(I)=1 ELSE B%(I)=0 'FLIP THE STATE
 3630 NEXT I
 3640 GOSUB 2850 UPDATE THE A AND B FIELD DISPLAYS
 3650 RETURN
 3660 REM **********************************
 3670 REM
                     RUN ITERATIONS OF THE NETWORK
 3690 IF ASYN=0 THEN GOTO 3940
 3700 REM **** PERFORM ASYNCHRONOUS UPDATE OF ASYN RANDOM NEURONS/FIELD ****
 3710 FOR CC=1 TO 10 ' LIMIT THE NUMBER OF ASYNCHRONOUS ITERATIONS
 3720
      FOR K=1 TO ASYN
 3730
         PIK=INT((AXSIZE*AYSIZE)*RND+1)
3740
         TSUM = 0
3750
         FOR J=1 TO BXSIZE*BYSIZE
                                            'UPDATE A FIELD NEURON
3760
           TSUM = TSUM + B%(J)*M%(PIK,J)
3770
         NEXT J
3780
         IF TSUM>O THEN A%(PIK)=1 'THRESHOLD FUNCTION
3790
         IF TSUM<0 THEN A%(PIK)=0
3800
       NEXT K
3810
       FOR K=1 TO ASYN
        PIK=INT((BXSIZE*BYSIZE)*RND+1)
3820
3830
        TSUM = 0
3840
        FOR I=1 TO AXSIZE*AYSIZE
                                           'UPDATE B FIELD NEURON
3850
          TSUM = TSUM + A%(I)*M%(I,PIK)
3860
        NEXT I
3870
        IF TSUM>O THEN B%(PIK)=1 'THRESHOLD FUNCTION
3880
        IF TSUM<0 THEN B%(PIK)=0
3890
      NEXT K
      GOSUB 2850 WANT TO WATCH THE PROGRESS
3900
3910 NEXT CC
3920 RETURN
                   ' COMPLETED ASYN ITERATION
3930 REM **** PERFORM SYNCHRONOUS UPDATE OF ALL NEURONS IN BOTH FIELDS ****
3940 FOR CC = 1 TO 2 'ONLY TWO ITERATIONS NEEDED
3950 FOR J=1 TO BXSIZE*BYSIZE
3960
        TSUM = 0
3970
        FOR I=1 TO AXSIZE*AYSIZE
                                           'UPDATE B FIELD NEURON
```

```
3980
          TSUM = TSUM + A\%(I)*M\%(I,J)
3990
        NEXT I
4000
        IF TSUM>O THEN B%(J)=1 'THRESHOLD FUNCTION
4010
        IF TSUM<0 THEN B%(J)=0
4020
      NEXT J
4030
      FOR I=1 TO AXSIZE*AYSIZE
4040
        TSIIM = 0
4050
        FOR J=1 TO BXSIZE*BYSIZE
                                            'UPDATE A FIELD NEURON
4060
          TSUM = TSUM + B\%(J)*M\%(I,J)
4070
4080
        IF TSUM>O THEN A%(I)=1 'THRESHOLD FUNCTION
4090
        IF TSUM<0 THEN A%(I)=0
4100 NEXT I
4110 GOSUB 2850
                     'UPDATE THE A AND B FIELD DISPLAYS
4120 NEXT CC ' ITERATION OF BOTH FIELDS
4130 RETURN
4140 RETURN
4150 REM *************************
4160 REM CLEAR THE A AND B FIELDS
4170 REM ***************************
4180 FOR I=1 TO 144
4190 A%(I)=0
4200 B%(I)=0
4210 NEXT I
4220 GOSUB 2850 'UPDATE THE A AND B FIELD DISPLAYS
4230 RETURN
4240 REM *********************************
4250 REM LOAD CORRELATION MATRIX M FROM DISK FILE
4260 REM *********************************
4270 CLS
4280 PRINT "
                     CURRENT MEMORY MATRIX FILES ON DISK"
4290 PRINT
4300 FILES "* . BAM"
4310 PRINT
4320 PRINT
4330 INPUT "
                    ENTER FILENAME TO LOAD MEMORY MATRIX: "; FILESPEC$
4340 IF FILESPEC$ = " " THEN RETURN
4350 IF INSTR(".", FILESPEC$) = 0 THEN FILESPEC$ = FILESPEC$ + ".BAM"
4360 OPEN FILESPEC$ FOR INPUT AS #1
4370 INPUT #1, AXSIZE
4380 INPUT #1, AYSIZE
4390 INPUT #1, BXSIZE
4400 INPUT #1, BYSIZE
4410 FOR J=1 TO BXSIZE*BYSIZE
4420 FOR I=1 TO AXSIZE*AYSIZE
      INPUT #1, M%(I,J)
4430
4440 NEXT I
4450 NEXT J
4460 CLOSE #1
4470 RETURN
4490 REM SAVE CORRELATION MATRIX M TO DISK FILE
4500 REM ********************************
4510 CLS
4520 PRINT "
                CURRENT MEMORY MATRIX FILES ON DISK"
4530 PRINT
4540 FILES "*. BAM"
4550 PRINT
4560 PRINT
4570 INPUT "
                   ENTER FILENAME TO SAVE MEMORY MATRIX: "; FILESPEC$
4580 IF FILESPEC$ = " " THEN RETURN
4590 IF INSTR(".", FILESPEC$) = 0 THEN FILESPEC$ = FILESPEC$ + ".BAM"
4600 OPEN FILESPEC$ FOR OUTPUT AS #1
4610 PRINT #1, AXSIZE
4620 PRINT #1, AYSIZE
4630 PRINT #1, BXSIZE
4640 PRINT #1, BYSIZE
4650 FOR J=1 TO BXSIZE*BYSIZE
4660 FOR I=1 TO AXSIZE*AYSIZE
4670
      PRINT #1, M%(I,J)
4680 NEXT I
4690 NEXT J
4700 CLOSE #1
4710 RETURN
4720 IF ERL=4540 THEN PRINT "NO FILES": RESUME 4550
4730 IF ERL=4300 THEN PRINT "NO FILES": RESUME 4310
```

4740 RESUME 1790 4750 CLOSE 4760 ON ERROR GOTO O 4770 END

SORTELEM. MOD From "Programming Project: Crafting Reusable Software in Modula-2" by Hanna Oktaba and René Berber, BYTE, September 1987.

```
IMPLEMENTATION MODULE SortElemType;
                  (* FROM FileDescriptor IMPORT FileDescr: *)
                 FROM InOut IMPORT in, ReadString, WriteString, WriteLn, Write;
                 FROM Storage IMPORT ALLOCATE;
                 FROM Strings IMPORT Length, Concat, Copy;
  CONST
                 EOS = OC;
                                                    (* End Of String *)
 TYPE
                 ElemType = POINTER TO FileDescr;
                 FileDescr = RECORD(* File descriptor *)
                           name : ARRAY [0..8] OF CHAR;
                           ext : ARRAY [0..3] OF CHAR;
                           size: ARRAY [0..7] OF CHAR;
                           date : ARRAY [0..8] OF CHAR;
                           time : ARRAY [0..6] OF CHAR
                     END:
 VAR
                 comp : PROCEDURE(ElemType, ElemType) : BOOLEAN;
 PROCEDURE compare (x, y: ElemType): BOOLEAN;
                          (* call the procedure currently *)
   RETURN comp(x,y)
                          (* assigned to "comp"*)
 END compare;
 PROCEDURE compName (r1, r2: ElemType): BOOLEAN;
 BEGIN
   RETURN StringComp(r1 .name, r2 .name)
 END compName;
 PROCEDURE compExt (r1, r2: ElemType): BOOLEAN;
   VAR temp1, temp2: ARRAY [0..12] OF CHAR;
 BEGIN (* compare by extension and then by name *)
   Concat(r1^.ext,".",temp1); Concat(temp1,r1^.name,temp1);
   Concat(r2^.ext,".",temp2); Concat(temp2,r2^.name,temp2);
   RETURN StringComp(temp1, temp2)
 END compExt;
 PROCEDURE select (option: CARDINAL);
 BEGIN
   CASE option OF
                                                    (* compare by:
       1 : comp: = compName
                                                         filenames
   2 : comp:= compExt
                                          (*extension
   ELSE comp: = compName
                                                    (* default
 END select;
PROCEDURE optionMenu;
   WriteString("options:"); WriteLn;
   WriteString("
                         1 to sort by filename"); WriteLn;
  WriteString("
                          2 to sort by extension"); WriteIn;
  WriteString(" the default is 1, any other is taken as 1");
  WriteLn; WriteLn
END optionMenu;
PROCEDURE ReadArray(VAR A: ARRAY OF ElemType): CARDINAL;
  VAR n, max : CARDINAL;
         temp : ARRAY [0..8] OF CHAR;
```

```
BEGIN
  n:= 0; max:= HIGH(A);
   ReadString(temp);
  WHILE (NOT in.eof) & (n < max) DO
         NEW(A[n]);
         Copy(temp,0,30,A[n] .name);
         ReadString(A[n] .ext);
ReadString(A[n] .size);
         ReadString(A[n] .date);
         ReadString(A[n] .time);
         ReadString(temp); INC(n)
   END;
  RETURN n
END ReadArray;
PROCEDURE WriteArray(A: ARRAY OF ElemType; n: CARDINAL);
  VAR 1 : CARDINAL;
BEGIN
  FOR i:= 0 TO n-1 DO
         WriteFString(A[i] .name,-11);
        WriteFString(A[i]^.ext,-6);
WriteFString(A[i]^.size,12);
WriteFString(A[i]^.date,10);
WriteFString(A[i]^.time,8); WriteLn
  END
END WriteArray;
PROCEDURE WriteFString (s: ARRAY OF CHAR; f: INTEGER);
         Write string "s" formated in a field of size f.
         IFf < 0 string is left justified
         If f > 0 string is right justified
         IF Length(s) > f string is truncated
         padding is done with blanks
*)
  VAR i, n: INTEGER;
          c : CHAR;
BEGIN
  n:= Length(s);
  IF f > O THEN FOR i:= 1 TO f-n DO Write('') END END;
  i:= 0:
  REPEAT c:= s[i]; Write(c); INC(i)
  UNTIL (i >= n) OR (i >= ABS(f));
  IF f < 0 THEN FOR i:= 1 TO -f-n DO Write('') END END
END WriteFString;
PROCEDURE StringComp (s1, s2: ARRAY OF CHAR): BOOLEAN;
(* returns s1 < s2 *)
  VAR 1, max : CARDINAL;
BEGIN
  i:= 0; max:= HIGH(s1);
  WHILE (i < max) & (s1[i] = s2[i]) DO
         IFs1[i] = EOS
         THEN RETURN FALSE
                                                          (*s1 = s2 *)
          ELSE INC(1)
        END
  END;
  RETURN s1[i] < s2[i]
END StringComp;
BEGIN
  comp: = compName
                                                       (* default *)
END SortElemType.
```

SORTTEST.MOD From "Programming Project: Crafting Reusable Software in Modula-2" by Hanna Oktaba and René Berber, BYTE, September 1987.

```
(*****

*
    Test of generic sorting routine

*
*****)
```

```
MODULE SortTest:
      FROM InOut IMPORT OpenInput, CloseInput, WriteString, WriteLn, ReadCard,
                       OpenOutput, CloseOutput;
      FROM Sort IMPORT Quort:
      FROM SortElemType IMPORT ElemType, select, optionMenu,
                              ReadArray, WriteArray;
      CONST
           N = 200;
VAR
  a : ARRAY [1..N] OF ElemType;
  n : CARDINAL:
                                         (* actual number of elements in "a" *)
  opt : CARDINAL;
BEGIN
  WriteString("Which file contains the data?");
  OpenInput("");
  n:= ReadArray(a);
  CloseInput;
  optionMenu;
  WriteString("Sort by ? ");
  ReadCard(opt); WriteLn;
  select(opt);
  Qsort(a,n);
  WriteIn; WriteString("Output file [ Esc for console ] ? ");
  OpenOutput("");
  WriteArray(a,n);
  CloseOutput
END SortTest.
SORTELEM.DEF From "Programming Project: Crafting Reusable Software in Modula-2" by Hanna Oktaba and René Berber, BYTE,
September 1987.
DEFINITION MODULE SortElemType;
(*
        This module is intended to describe the elements to be sorted
        as an abstract data type.
 *)
        EXPORT QUALIFIED ElemType, compare, select, optionMenu,
```

ReadArray, WriteArray; TYPE ElemType; (* pointer to data element *) PROCEDURE compare (x, y: ElemType): BOOLEAN; (* compare(x,y) implements: x < y defined as NOT (y <= x), for ascending order; and if descending order is desired compare(x,y) should implement: x > y defined as NOT (x <= y); where "<=" denotes a binary relation that must satisfy the total order properties: 1. x <= x 2. $x \le y$ AND $y \le x = y$ 3. x <= y AND y <= z ==> x <= z 4. $x \le y$ OR $y \le x$ for every x, yPROCEDURE select (option: CARDINAL); (* input: - a number denoting the requested option output: - the exported compare procedure gets assigned to one of the comparison procedures. - the option should be valid; otherwise a default may be used *) PROCEDURE optionMenu; (* output: - displays on the screen the available options. *)

```
PROCEDURE ReadArray(VAR A: ARRAY OF ElemType): CARDINAL;
         (* input:
                                     - an array of pointers, declared by the user module.
           output:
                             - the array is filled with pointers to the memory used by the elements
                                   read, and the number of them is returned.
           errors:
                             - can run out of memory.
                           NOTE: If the file contains more elements than those that can be stored in the array, they are ignored.
PROCEDURE WriteArray(A: ARRAY OF ElemType; n: CARDINAL);
         (* input:
                                      - an array of pointers, and the number of elements.
           output:
                             - the elements are written to current output of InOut.
END SortElemType.
           From "Programming Project: Crafting Reusable Software in Modula-2" by Hanna Oktaba and René Berber, BYTE, September
SORT.MOD
1987.
IMPLEMENTATION MODULE Sort;
          FROM SortElemType IMPORT ElemType, compare;
PROCEDURE Qsort (VAR A: ARRAY OF ElemType; N: CARDINAL);
  PROCEDURE sort (1, r: INTEGER);
                                            (* N. Wirth '86 *)
         VAR i, j : INTEGER;
               x, w : ElemType;
  BEGIN
         i:=1; j:=r;
         x:= A[(1+r) DIV 2];
         REPEAT
           WHILE compare(A[i],x) DO INC(i) END;
           WHILE compare(x, A[j]) DO DEC(j) END;
           IF i <= j
             THEN w := A[i]; A[i] := A[j]; A[j] := w;
               INC(i); DEC(j)
          END:
        UNTIL i > j;
         IF 1 < j THEN sort(1,j) END;
         IF i < r THEN sort(i,r) END
  END sort;
BEGIN
         IF N > HIGH(A)+1 THEN N: = HIGH(A)+1 END;
        sort(0, N-1)
END Qsort;
END Sort.
ADD-PATT.C From "Constructing an Associative Memory" by Bart Kosco, BYTE, September 1987.
ADD-PATT ADDS A PATTERN TO THE BAM MATRIX XTY
COPYRIGHT (C) 1987, JOELS. DAVIS. AFTER BYTE PUBLICATION, APPROVED FOR
NONCOMMERCIAL USE ONLY.
#include "STRUCT.H"
#include "EXT.H"
add_patt(an,bn)
int *an, *bn;
      int i,j;
```

```
MAKE VECTORS BIPOLAR */
      for(i=0;i<*an;i++){
                if(a[i]<=0){
                  x[i] = -1;
                }else{
                  x[i] = 1;
      for(j=0;j<*bn;j++){
               if(b[j]<=0){
                  y[j] = -1;
                }else{
                  y[j] = 1;
      UPDATE XTY */
      for(i=0;i<*an;i++){
               for(j=0;j<*bn;j++){
                  xty[i][j] += x[i]*y[j];
      SIGNAL COMPLETION */
      putchar(BELL);
          From "Programming Project: Crafting Reusable Software in Modula-2" by Hanna Oktaba and René Berber, BYTE, September
SORT.DEF
1987.
DEFINITION MODULE Sort;
         FROM SortElemType IMPORT ElemType, compare;
(* Module SortElemType is used to define the kind of element to be sorted *)
         EXPORT QUALIFIED Qsort;
PROCEDURE Qsort (VAR A: ARRAY OF ElemType; N: CARDINAL);
(* input:
                         - array of pointers to elements, and number of elements
                             [N<=HIGH(A)+1].
 * output:
                  -the array of pointers is rearranged so they point to the
                             elements in sorted order.
 * requires that ElemType has a total order relation named "compare".
 *)
END Sort.
README.BAM From "Constructing an Associative Memory" by Bart Kosco, BYTE, September 1987.
TO RUN BAM:
           FILES BAM.COM, BAM.PAS, XFACE.INC, PAT1(NOT NECESSARY)
          README.BAM - this file
          > BAM
           ** RUNS THE PROGRAM **
           SYSTEM REQUIREMENTS: COLOR MONITOR (CGA) OR Mono Graphics
                     TURBO PASCAL (FOR EDITING)
```

STATUS LINE: PRESS A, S, H, OR Q

```
ANSWER YES/NO QUESTIONS WITH Y OR N < RETURN>.
           PAT1 is a sample pattern file.
code 15,712 bytes
data 49,424 bytes
FFI:
           Dr. Rod Taber
           General Dynamics
           Electronics Division Mail Zone 7202-K
           Box 85310
           San Diego, CA 92138
Mail without Mail Zone takes 3 months.
AWAIT.C From "Constructing an Associative Memory" by Bart Kosco, BYTE, September 1987.
AWAIT ASKS THE USER FOR ANOTHER KEY ENTRY AFTER N RETURNS
await(im)
int im;
int i;
     for(i=0;i<im;i++){
       printf("\n");
     printf("Press any key to continue: ");
     getchar();
ASYNCH.C From "Constructing an Associative Memory" by Bart Kosco, BYTE, September 1987.
ASYNCH ASYNCHRONOUSLY UPDATES THE BAM
COPYRIGHT (C) 1987, JOEL S. DAVIS. AFTER BYTE PUBLICATION, APPROVED FOR
NONCOMMERCIAL USE ONLY.
*/
#include "STRUCT.H"
#include "EXT.H"
asynch(axn,ayn,bxn,byn,an,bn,n_asyn)
int *axn, *ayn, *bxn, *byn, *an, *bn, n_asyn;
int i, j, k, l, m, n, index, iter, sum;
int seed;
char buf[9];
double x, frand();
/* GET A RANDOM NUMBER SEED FROM THE SYSTEM CLOCK */
/* READ THE CLOCK */
    times(buf);
/* GENERATE THE SEED INTEGER */
    seed = (int) (buf[7] - '0');
    seed += seed + 10 * ((int) ((buf[6] - '0')));
```

seed += seed + 60 * ((int) ((buf[4] - '0')));
seed += seed + 600 * ((int) ((buf[3] - '0')));

srand(seed);

```
/* GO THROUGH 10 ITERATIONS */
     iter = 10:
     for(k=0;k<iter;k++){
          UPDATE N_ASYN NEURONS */
          for(n=0;n< n_asyn;n++){
                 sum = 0;
                 PROCESS B VECTOR NEURON */
 /*
                 WHICH NEURON DO I UPDATE THIS TIME ? */
                 x = (double) *bn;
                 x *= frand();
                 x += 0.5;
                 j = (int) x;
/*
                 APPLY ALL A VECTOR INPUTS */
                 for(i=0;i<*an;i++){
                    sum += a[i] * xty[i][j];
/*
                 NOW DO THRESHOLDING */
                 if(sum>0){
                    b[j] = 1;
                 }else if(sum<0){
                   b[j] = 0;
                 }else{
/ *
                    RETAIN VALUE */
                    continue;
         vec_show(axn,ayn,bxn,byn);
         NOW IN REVERSE TO PROCESS A VECTOR NEURONS */
/*
         UPDATE N_ASYN NEURONS */
         for(n=0;n<n_asyn;n++){
                sum = 0;
                PROCESS B VECTOR NEURON */
/ *
                WHICH NEURON DO I UPDATE THIS TIME ? */
                x = (double) *an;
                x *= frand();
                x += 0.5;
                i = (int) x;
                FOR EACH A VECTOR NEURON, APPLY ALL B VECTOR INPUTS */
                for(j=0;j<*bn;j++){}
                   sum += b[j] * xty[i][j];
                NOW DO THRESHOLDING */
                if(sum>0){
                   a[i] = 1;
                }else if(sum<0){
                   a[i] = 0;
               }else{
               RETAIN VALUE */
                   continue;
        vec_show(axn,ayn,bxn,byn);
```

SIEVE386.ASM Accompanies "The Kaypro 386" by Ray Duncan, BYTE, September 1987.

```
Title "Eratosthenes Sieve for 80386"
Name Sieve
Page 50,132
```

```
; Eratosthenes Sieve for 80386 32-bit protected mode
  ; Implemented by Ray Duncan, April 1987
  ; After Gilbreath, BYTE, September 1981 and January 1983
  ; Here is the MAKE file for this program:
  ; sieve386.obj : sieve386.asm
  ; 386asm sieve386
 ; sieve386.exe : sieve386.obj
 ; 3861ink sieve386 start386 -exe sieve386 -map sieve386
 ; To run the program with Phar Lap DOS EXTENDER:
 ; -C>RUN386 SIEVE386
 niter
           eau
                   100
                                          ; number of iterations
 asize
           equ
                   8190
                                          ; size of array "flags"
 or
           equ
                   Odh
                                          ; ASCII carriage return
 1f
           equ
                   Oah
                                          ; ASCII linefeed
 stdin
                   0
           eau
                                          ; handle for standard input
 stdout
           equ
                   1
                                          ; handle for standard output
 TEXT
           segment para public use32 'CODE'
           assume cs:_TEXT, ds:_DATA, es:_DATA
           public _start_
                                          ; magic name for RUN386 entry
 _start_
          proc
                  near
          xor
                  edx, edx
                                          ; convert number of iterations
          mov
                  eax, niter
                                          ; for output
          mov
                  ecx,10
                  esi, offset msgla+3
          mov
          call
                  binasc
          mov
                  edx, offset msg1
                                         ; display message
          mov
                  ecx, msg1_len
                                          ; "Starting N iterations of Sieve"
          call
                  pmsg
          call
                  getmsec
                                         ; get current time in msec
          push
                  eax
                                         ; and save it ...
          mov
                  counter, niter
                                         ; initialize iterations counter
sievel:
                                         ; a sieve iteration starts here...
          mov
                  edi, offset flags
                                         ; initialize flags array
          mov
                  ecx, asize
                                         ; to all bytes = TRUE
          mov
                  al,1
          cld
          rep stosb
          xor
                  esi, esi
                                         ; ESI = index to flags array
                  edi, edi
          xor
                                         ; EDI = primes counter
sieve2:
                                         ; main loop of sieve
                  byte ptr flags[esi],1; is this a prime?
          test
          jnz
                  short sieve4
                                         ; jump if prime
sieve3:
          inc
                 esi
                                         ; bump to next slot in "flags"
          cmp
                  esi, asize
          jle
                 sieve2
                                         ; loop until array exhausted
          dec
                 counter
                                         ; count off sieve iterations
          jnz
                 sieve1
                                         ; jump, another iteration needed.
                 sieve7
          .jmp
                                         ; jump, all iterations finished.
sieve4:
                                         ; prime found, zap its multiples
         mov
                 ebx, esi
                                         ; copy i
         mov
                 edx, ebx
                                         ; EDX = prime = i + i + 3
         add
                 edx, edx
         add
                 edx,3
```

```
xor
                    al,al
            jmp
                   short sieve6
                   byte ptr flags[ebx], al; zero this multiple
  sieve5:
           mov
  sieve6:
           add
                   ebx, edx
                                           ; find next multiple of prime
                   ebx, asize
           cmp
                                           ; have we exhausted the array?
           ile
                   sieve5
                                           ; not yet, zap it
           inc
                   edi
                                           ; count primes and try next
           jmp
                   sieve3
 sieve7:
           call
                   getmsec
                                          ; all done, get current time
           push
                   eax
           mov
                   eax, edi
                                          ; convert number of primes
                   edx,0
           mov
                                          ; found on last iteration
           mov
                   ecx,10
           mov
                   esi, offset msg2a+4
           call
                   binasc
           mov
                   edx, offset msg2
                                          ; display "Number of primes: "
           mov
                   ecx, msg2_len
           call
                   pmsg
           pop
                   eax
                                          ; calculate total elapsed msec.
           pop
                   ebx
           sub
                   eax, ebx
           mov
                   edx,0
                                          ; divide by number of iterations
           mov
                  ecx, niter
                                          ; to get msec per iteration
           idiv
                   ecx
           mov
                  edx,0
                                          ; convert msec to ASCII
           mov
                  ecx,10
           mov
                  esi, offset msg3a+4
           call
                  binasc
          mov
                  edx, offset msg3
                                          ; display "Elapsed time: "
          mov
                  ecx, msg3_len
          call
                  pmsg
          mov
                  ax.04C00h
                                          ; final exit, return code = 0
          int
                  21H
 _start_
          endp
 getmsec proc
                  near
                                         ; Return EAX = current time in msec.
          mov
                  ah,2ch
                                         ; read time
          int
                  21h
          movzx
                  eax, ch
                                         ; EAX := hours
          imul
                  eax,60
                                         ; hours -> minutes
          and
                  ecx, Offh
                                         ; isolate system minutes
          add
                  eax,ecx
                                         ; and find total minutes
          imul
                  eax,60
                                         ; minutes -> seconds
          movzx
                 ecx, dh
                                         ; isolate system seconds
          add
                  eax, ecx
                                         ; and find total seconds
          and
                  edx, Offh
                                         ; isolate hundredths
          imul.
                  eax,100
                                         ; seconds -> hundredths
          add
                  eax, edx
                                         ; find total hundredths
          imul
                  eax,10
                                         ; hundredths -> msec
          ret
getmsec endp
; BINASC: Convert 64-bit binary value to ASCII string.
; Call with EDX: EAX = signed 64-bit value
           ECX
                   = radix
           DS:ESI = last byte of area to store resulting string
                     (make sure enough room is available to store
                     the string in the radix you have selected.)
; Destroys EAX, EBX, ECX, EDX, and ESI.
```

```
binasc proc
                near
                                         ; convert EDX: EAX to ASCII.
                byte ptr [esi],'0'
        mov
                                        ; force storage of at least one digit.
        or
                edx, edx
                                        ; test sign of 64-bit value,
        pushf
                                        ; and save sign on stack.
        jns
                bin1
                                        ; jump if it was positive.
        not
                edx
                                        ; negative, take 2's complement
        not
                                        ; of the value.
        add
                eax.1
        adc
                                        ; divide 64-bit value by the radix
 bin1:
                                        ; to extract next digit for the
                                        ; forming string.
        mov
                ebx, eax
                                        ; is the value zero yet?
        or
                ebx,edx
        jz
                bin3
                                        ; yes, we are done converting.
                divide
        call
                                        ; no, divide by radix.
        add
                b1,'0'
                                        ; convert remainder to ASCII digit.
                b1, '9'
        cmp
                                        ; might be converting hex ASCII,
        jle
                bin2
                                        ; jump if in range 0-9,
        add
                bl,'A'-'9'-1
                                        ; correct it if in range A-F.
 bin2:
        mov
                [esi],bl
                                        ; store this character into string.
        dec
                esi
                                        ; back up through string,
        jmp
                bin1
                                        ; and do it again.
 bin3:
                                        ; restore sign flag,
        popf
                                        ; was original value negative?
        jns
                bin4
                                        ; no, jump
                byte ptr [esi],'-'
        mov
                                        ; yes, store sign into output string.
 bin4: ret
                                        : back to caller.
 binasc endp
; General-purpose 64-bit by 32-bit unsigned divide.
; This must be used instead of the plain machine unsigned divide
; for cases where the quotient may overflow 32 bits. If called with
; zero divisor, this routine returns the dividend unchanged and gives
; no warning.
; Call with EDX: EAX = 64-bit dividend
           ECX
                   = divisor
; Returns EDX: EAX = quotient
           EBX
                   = remainder
           ECX
                    = divisor (unchanged)
divide
         proc
                 near
                                         ; Divide EDX: EAX by ECX
          jecxz
                 div1
                                         ; exit if divide by zero
          push
                 eax
                                         ; 0:dividend_upper/divisor
         mov
                 eax, edx
         xor
                 edx, edx
         div
                 ecx
         mov
                 ebx, eax
                                         ; EBX = quotient1
         pop
                 eax
                                         ; remainder1:dividend_lower/divisor
         div
                 ecx
         xchg
                 ebx,edx
                                        ; EDX: EAX = quotient1: quotient2
div1:
                                         ; EBX = remainder2
         ret
divide
         endp
pmsg
         proc
                 near
                                         ; print a message on std output
                                         ; call with DS: EDX = address
                                                   ECX = length
          mov
                 ah,40h
                 bx.stdout
         mov
         int
                 21h
         ret
pmsg
         endp
_TEXT
         ends
_DATA
         segment para public use32 'DATA'
```

```
db
flags
                asize+1 dup (?)
counter dd
                                       ; sieve iteration counter
         db
                cr, lf, 'Starting '
             ' iterations of Sieve...', cr, lf
       db
msg1a
msg1_len equ $-msg1
msg2
         db
                cr, lf, 'Primes found: '
msg2a
         db
                    ',cr,lf
msg2_len equ $-msg2
         db
msg3
                cr, lf, 'Elapsed time: '
                   msec. per iteration', cr, lf
msg3a
         db
msg3_len equ $-msg3
_DATA
         ends
_STACK
         segment byte stack use32 'stack'
         db
                4096 dup (?)
STACK
         ends
         end
```

GET_MTX.C From "Constructing an Associative Memory" by Bart Kosco, BYTE, September 1987.

```
GET_MTX GETS THE VALUES FOR AN XTY MEMORY MATRIX FROM MASS STORAGE
COPYRIGHT (C) 1987, JOELS. DAVIS. AFTER BYTE PUBLICATION, APPROVED FOR NONCOMMERCIAL USE ONLY.
#include <c:\cware\stdio.h>
#include <struct.h>
#include <ext.h>
get_mtx(axn,ayn,bxn,byn,an,bn)
int *axn, *ayn, *bxn, *byn, *an, *bn;
   int i, j, total;
   FILE *f3, *fopen();
   char fname[40];
   for(;;){
      for(i=0;i<40;i++) fname[i] = 0;
      cls0();
      printf("\n\n\nPlease enter name of file containing memory values");
      printf( "\nyou wish to restore: ");
      scanf("%s", fname);
       OPEN INPUT FILE */
      if((f3=fopen(fname,"r"))==NULL){
        printf("\nCan't open INPUT FILE %s\n", fname);
        printf("\nTry again! Hit [ENTER] to continue...");
        getchar();
      }else{
        break;
  }
   CLEAR THE BAM */
  clearbam();
   READ HORIZONTAL AND VERTICAL DIMENSIONS OF A AND B ARRAYS */
  fscanf(f3, "%d %d %d %d", axn, ayn, bxn, byn);
  printf("\n\n Reading A array %d x %d and B array %d x %d",
          *axn, *ayn, *bxn, *byn);
  *an = (*axn) * (*ayn);
  *bn = (*bxn) * (*byn);
  total = (*an) * (*bn);
```

```
printf("\n%dx%d xTy -> %d lines to be read!",*an,*bn,total);
printf("\nDepending upon your machine, this could take a minute or so...");

for(i=0;i<*an;i++){
   for(j=0;j<*bn;j++){
    fscanf(f3,"%d",&xty[i][j]);
   }
}
fclose(f3);</pre>
```

PAT1 From "Constructing an Associative Memory" by Bart Kosco, BYTE, September 1987.

EXT.H From "Constructing an Associative Memory" by Bart Kosco, BYTE, September 1987.

```
/*
BAM PARAMETER ARRAYS

COPYRIGHT (C) 1987, JOELS. DAVIS. AFTER BYTE PUBLICATION, APPROVED FOR NONCOMMERCIAL USE ONLY.

*/

extern int a[DIMEN1],b[DIMEN2],x[DIMEN1],y[DIMEN2],xty[DIMEN1][DIMEN2];

extern int newvec1[DIMEN1],newvec2[DIMEN2];
```

BAM.C From "Constructing an Associative Memory" by Bart Kosco, BYTE, September 1987.

```
/*
PROGRAM BAM

BY
JOELS. DAVIS
1310 CONSTITUTION COURT, NE
ALBUQUERQUE, NM 87112
```

THIS PROGRAM IS A SIMPLE IMPLEMENTATION OF THE BIDIRECTIONAL CORRELATION MEMORY ALGORITHM DEVELOPED BY BART KOSKO. THE SOURCE CODE IS WRITTEN FOR DESMET C BY CWARE RUNNING UNDER MS-DOS.

IT COMBINES FEATURES FROM THE DUANE DESIENO BAM PROGRAM, WRITTEN IN BASIC, AND MY OWN ASSOC AND BASSOC PROGRAMS, WRITTEN IN C.

THIS VERSION IS DATED: 7-AUGUST-1987

COPYRIGHT (C) 1987, JOEL S. DAVIS. AFTER BYTE PUBLICATION, APPROVED FOR NONCOMMERCIAL USE ONLY.

```
#include <\cware\stdio.h>
#include < \cware \math.h>
#include <struct.h>
int a[DIMEN1],b[DIMEN2],x[DIMEN1],y[DIMEN2],xty[DIMEN1][DIMEN2];
int newvec1[DIMEN1],newvec2[DIMEN2];
main(argc,argv)
int argc;
char **argv;
int i, j, k, l, m, n, d2;
int op;
int axn, ayn, bxn, byn, n_asyn, pct_noise, an, bn;
         INITIALIZE DIMENSIONS */
         clearbam();
         initial(&axn,&ayn,&bxn,&byn,&an,&bn,&n_asyn,&pct_noise);
         INTRODUCTION */
         intro();
         PRESENT MAIN MENU */
         for(;;){
               op = mainmenu();
               if(op==1){
                    NEW PROGRAM PARAMETERS */
                   new_parm(&axn,&ayn,&bxn,&byn,&an,&bn,&n_asyn,&pct_noise);
               }else if(op==2){
                 CLEAR VECTORS AND BAM */
                 clearbam();
               }else if(op==3){
                 LOAD THE MEMORY MATRIX */
                 get_mtx(&axn,&ayn,&bxn,&byn,&an,&bn);
               }else if(op==4){
                 SAVE THE MEMORY MATRIX */
                 save_mtx(&axn,&ayn,&bxn,&byn,&an,&bn);
              }else if(op==5){
                 OPERATE ON MEMORY AND RUN BAM */
                 run_bam(&axn,&ayn,&bxn,&byn,&an,&bn,n_asyn,pct_noise);
              }else if(op==6){
                 QUIT */
                 break;
        }
```

CLEARBAM.C From "Constructing an Associative Memory" by Bart Kosco, BYTE, September 1987.

```
/*
CLEARBAM CLEARS ALL BAM VECTORS AND THE XTY MATRIX

COPYRIGHT (C) 1987, JOEL S. DAVIS. AFTER BYTE PUBLICATION, APPROVED FOR NONCOMMERCIAL USE ONLY.

*/
#include <STRUCT.H>
#include <EXT.H>

clearbam()
{
int d2;

/* SET MATRICES TO ZERO */
    zero_1d(a, DIMEN1);
    zero_1d(b, DIMEN2);
```

```
zero_ld(x,DIMEN1);
zero_ld(y,DIMEN2);
zero_ld(newvec1,DIMEN1);
zero_ld(newvec2,DIMEN2);

/* NOW 2-DIMENSIONAL VECTORS, WHICH WILL BE TREATED LIKE 1-D VECTORS */
d2 = DIMEN1 * DIMEN2;
zero_ld(xty,d2);
}
```

SIEVE86.ASM Accompanies "The Kaypro 386" by Ray Duncan, BYTE, September 1987.

```
Title "Eratosthenes Sieve for 80x86 Real Mode"
                  Sieve
           Name
           Page
                  50,132
 ; Eratosthenes Sieve for 80x86 Real Mode
 ; Implemented by Ray Duncan, April 1987
 ; After Gilbreath, BYTE, September 1981 and January 1983
                    100
 niter
           equ
                                               ; number of iterations
 asize
          equ
                    8190
                                               ; size of array "flags"
 cr
          equ
                    0dh
                                               ; ASCII carriage return
 1f
          equ
                    Oah
                                               ; ASCII linefeed
 stdin
                    0
          equ
                                               ; handle for standard input
stdout
                    1
                                               ; handle for standard output
 _TEXT
          segment para public 'CODE'
          assume cs:_TEXT,ds:_DATA,es:_DATA
sieve
          proc
                    near
          mov
                    ax, seg_DATA
          mov
                    ds,ax
          mov
                    es,ax
                    dx,0
          mov
                                               ; convert number of iterations
                    ax, niter
                    cx,10
          mov
          mov
                    si, offset msgla+3
          call
                    binasc
                    dx, offset msg1
          mov
                                               ; display message
          mov
                    cx, msg1_len
                                               ; "Starting N iterations of Sieve"
          call
                    pmsg
          call
                    getmsec
                                               ; get current time in msec
          push
                    dx
                                               ; and save it ...
          push
                    ax
          mov
                    counter, niter
                                               ; initialize iterations counter
sievel:
                                              ; a sieve iteration starts here...
                    di, offset flags
          mov
                                              ; initialize flags array
          mov
                    cx, asize
                                              ; to all bytes = TRUE
          mov
                    al.1
          cld
          rep stosb
                    si,si
         xor
                                              ; SI = index to flags array
          xor
                    di, di
                                              ; DI = primes counter
sieve2:
                                              ; main loop of sieve
          test
                   byte ptr flags[si],1
                                              ; is this a prime?
         jnz
                   short sieve4
                                              ; jump if prime
```

```
sieve3:
            inc
                      si
                                                 ; bump to next slot in "flags"
            cmp
                      si, asize
                                                 ; are we done?
            jle
                      sieve2
                                                 ; jump to test another
            dec
                      word ptr counter
                                                 ; more iterations?
            jnz
                      sieve1
                                                 ; jump, another iteration needed
            jmp
                      sieve7
  sieve4:
                                                 ; prime found, zap its multiples
            mov
                      bx,si
                                                 ; copy i
            mov
                      dx, bx
                                                 ; DX = prime = i + i + 3
            add
                      dx, dx
            add
                      dx,3
            xor
                      al, al
            jmp
                      short sieve6
  sieve5: mov
                      byte ptr flags[bx],al
                                                ; zero this multiple
  sieve6:
           add
                      bx, dx
                                                ; find next multiple of prime
            cmp
                      bx, asize
                                                ; have we exhausted the array?
            jle
                      sieve5
                                                ; not yet, zap it
           inc
                     di
                                                ; count primes and try next
           jmp
                     sieve3
 sieve7:
                                                ; done with all iterations ...
           call
                     getmsec
                                                ; get current time
           push
                     dx
                                                ; and save it...
           push
                     ax
           mov
                     ax,d1
                                                ; convert number of primes
           mov
                     dx,0
           mov
                     ex,10
           mov
                     si, offset msg2a+4
           call
                     binase
           mov
                     dx, offset msg2
                                                ; display "Number of primes:"
           mov
                     cx,msg2_len
           call
                     pmsg
          pop
                     ax
                                                ; stop time: low word
                     dx
          pop
                                                           high word
          pop
                     bx
                                               ; start time: low word
          pop
                    CX
                                                             high word
          sub
                    ax, bx
                                               ; DX:AX = stop - start
          sbb
                    dx, ex
          mov
                    ex, niter
                                               ; divide by number of iterations
          idiv
                    ox
                    dx,0
          mov
                                               ; convert msec to ASCII
          mov
                    ex,10
          mov
                    si,offset msg3a+4
          call
                    binase
                    dx, offset msg3
          mov
                                              ; display "Elapsed time:"
          mov
                    ox,msg3_len
          call
                    pmag
                    ax,04000h
          mov
                                               ; exit to DOS with
                    21H
                                               ; return code = 0
sieve
getmsec
         proc
                   near
                                              ; DX:AX := current time in msec.
         mov
                   ah,2ch
                                              ; read time from MS-DOS
         int
                   21h
         push
                                              ; save seconds, hundredths
         mov
                   al, ch
                                              ; AX := hours
         chw
         mov
                   bx,60
                                              ; hours -> minutes
         1mul
                   bx
                   ch, ch
                                              ; isolate system minutes
```

```
add
                     ax, cx
                                                 ; and find total minutes
          mov
                     bx.60
                                                : minutes -> seconds
          imul
                     bx
                     cx
                                                ; recover seconds, hundredths
          pop
          mov
                     bl,ch
                                                ; get seconds
          xor
                     bh, bh
                     ax, bx
                                                ; find total seconds
          add
                     dx,0
          adc
                                                ; carry if necessary
                     ch, ch
          xor
                                                ; save centisec.
          mov
                     bp, cx
                                                ; total seconds * 100 the hard way
                     bx,ax
                                                ; double multiply * 10
          mov
                     cx,dx
          mov
          add
                     ax,ax
          ade
                     dx, dx
          add
                     ax,ax
                                                ; * 4
                     dx, dx
          ade
          add
                     ax, bx
                                                ; * 5
          ade
                     dx, cx
          add
                     ax, ax
                                                ; * 10
                     dx, dx
          adc
                                                ; double multiply * 10
          mov
                     bx.ax
          mov
                     cx,dx
                     ax,ax
          add
                                                ; * 2
          adc
                     dx, dx
          add
                     ax,ax
                                                ; * 4
                     dx, dx
          adc
          add
                     ax,bx
                                                ; * 5
          adc
                     dx, cx
          add
                     ax,ax
                                                ; * 10
                     dx, dx
          adc
          add
                     ax, bp
                                                ; add in hundredths of seconds
          adc
                     dx,0
                                                ; now convert total to msec.
                                                ; double multiply * 10
                     bx, ax
          mov
                     cx, dx
          add
                     ax,ax
                                                ; * 2
                     dx, dx
          adc
                                                ; * 4
          add
                     ax, ax
          adc
                     dx, dx
          add
                     ax, bx
                                                ; * 5
          adc
                     dx, cx
          add
                     ax, ax
                                                ; * 10
          adc
                     dx, dx
                                                ; return DX: AX = msec.
          ret
getmsec endp
; BINASC: Convert 32-bit binary value to ASCII string.
; Call with DX: AX = signed 32-bit value
           CX
                 = radix
                  = last byte of area to store resulting string
                  (make sure enough room is available to store the string in the radix you have selected.)
; Destroys AX, BX, CX, DX, and SI.
binasc
                                                ; convert DX: AX to ASCII.
          proc
          mov
                    byte ptr [si],'0'
                                               ; force storage of at least one digit.
                                               ; test sign of 32-bit value,
          or
                    dx, dx
          pushf
                                               ; and save sign on stack.
          jns
                    bin1
                                               ; jump if it was positive.
          not
                    dx
                                               ; negative, take 2's complement
          not
                    ax
                                               ; of the value.
          add
                    ax,1
          adc
                    dx,0
bin1:
                                               ; divide 32-bit value by radix
                                               ; to extract next digit for the
                                               ; forming string.
          mov
                    bx,ax
                                               ; is the value zero yet?
          or
                    bx,dx
```

```
jz
                      bin3
                                                 ; yes, we are done converting.
            call
                      divide
                                                 ; no, divide by radix.
            add
                      b1,'0'
                                                 ; convert remainder to ASCII digit.
                      b1, 191
            cmp
                                                 ; might be converting to hex ASCII,
            jle
                      bin2
                                                 ; jump if in range 0-9,
            add
                      bl. 'A'-19'-1
                                                 ; correct it if in range A-F.
  bin2:
            mov
                      [si],bl
                                                 ; store this character into string.
            dec
                      81
                                                 ; back up through string,
            jmp
                      bin1
                                                 ; and do it again.
  bin3:
                                                 ; restore sign flag,
            popf
                                                 ; was original value negative?
            jns
                      bin4
                                                 ; no, jump
            mov
                      byte ptr [si], '-'
                                                 ; yes, store sign into output.
  bin4:
            ret
                                                 ; back to caller.
  binasc
            endp
  ; General-purpose 32-bit by 16-bit unsigned divide.
  ; This must be used instead of the plain machine unsigned divide for cases where the quotient may overflow 16 bits
  ; (for example, dividing 100,000 by 2). If called with a zero divisor, this routine returns the dividend unchanged
  ; and gives no warning.
   Call with DX: AX = 32-bit dividend
             CX
                 = divisor
  : Returns
            DX: AX = quotient
             BX
                  = remainder
             CX
                   = divisor (unchanged)
 divide
           proc
                     near
                                                ; Divide DX: AX by CX
           jexz
                     div1
                                                ; exit if divide by zero
           push
                     ax
                                                ; 0:dividend_upper/divisor
           mov
                     ax, dx
           xor
                     dx, dx
           div
                     ox
           mov
                     bx,ax
                                                ; BX = quotient1
           pop
                     ax
                                                ; remainder1:dividend_lower/divisor
           div
                     OX
           xchg
                     bx, dx
                                                ; DX: AX = quotient1: quotient2
 div1:
           ret
                                                ; BX = remainder2
 divide
           endp
 pmsg
           proc
                     near
                                                ; print a message on std output
                                                ; call with DS: EDX = address
                                                           ECX
                                                                  = length
          mov
                     ah,40h
          mov
                     bx,stdout
          int
                    21h
          ret
 pmsg
          endp
 _TEXT
          ends
_DATA
          segment para public 'DATA'
flags
          db
                    asize+1 dup (?)
counter
          dw
msg1
          db
                    er, 1f, 'Starting '
msg1a
          db
                            iterations of Sieve...', cr, lf
msg1_len equ $-mag1
msg2
          db
                    er, 1f, 'Primes found: '
msg2a
          db
                            ', or, lf
msg2_len equ $-mag2
msg3
                    er, 1f, 'Elapsed time: '
         db
msg3a
                           msec. per iteration', cr, lf
msg3_len equ $-mag3
```

```
_DATA ends
_STACK segment byte stack 'stack'
db 4096 dup (?)
_STACK ends
end sieve
```

TURBFLOP.PRO Contributed by Alex Lane. From "ALS Prolog," BYTE, September 1987.

```
/* Floating-Point Test Program
/* written in Turbo Prolog
/* 6-10-87 a.lane
/* result for 5000 repetitions:
     30 seconds
predicates
        float_point
        time_1(integer)
        cycle(integer, real, real, real)
        calc(real, real, real, real)
goal
        float_point.
clauses
        float_point :-
                write( "Enter number of repetitions: " ),
                readint(Iters),
                 time_1(Start),
                cycle(Iters, 1.0, 2.71828, 3.14159),
                 time_1(Finish),
                Overall = Finish - Start,
                 write( "Time is " ),
                write(Overall),nl.
        calc(C,CF,A,B):-
                C1 = C * A,
                C2 = C1 * B,
                C3 = C2 / A,
                CF = C3 / B.
        cycle(0,C,_,_):-
write("Cis"),
                write(C),nl.
        cycle(N,C,A,B):-
                calc(C,CF,A,B),
                N1 = N - 1,
                cycle(N1,CF,A,B).
        time_1( Time ) :-
                /* quick and dirty; won't work across midnight */
                /* and ignores hundreths of a second
                time(H,M,S,_),
                Time = S + 60 * (M + 60 * H).
```

KAREX2.BAS Accompanies the article "Karmarkar's Algorithm" by Andrew M. Rockett and John C. Stevenson, BYTE, September 1987.

```
106 '
 107 ' by Andrew M. Rockett and John C. Stevenson
 108 '
 109 ' This program was written by Andrew M. Rockett
 110 '
 111 ' -
 200 '
 202 'N is the number of unknowns and K is the number of equations
 204 1
 206 N = 8 : K = 4
 208 1
 210 K1 = K + 1 : K2 = 2*K1
 212 DIM AO(N), XOLD(N), XNEW(N), CC(N), CP(N),
     A(K,N), B(K1,N), B1(K1,K2), B2(N,K1), B3(N,N)
 214 FOR C = 1 TO N : AO(C) = 1/N : XNEW(C) = AO(C) :
 216 '
 218 ' T is the tolerance
 220 '
 222 T = .001
 224 1
 226 ' ALPHA is usually set equal to 1/4 ...
 228 1
 230 ALPHA = .25
232 1
 234 ITERATION = 0
236 1
 238 ' Data for constraint matrix A
240 1
242 DATA 1, 0, 1, 0, 0, 0, 1, -3
244 DATA 1, 0, 0, -1, 0, 0, 2, -2
246 DATA 0, 1, 0, 0, 1, 0, 3, -5
248 DATA 0, 1, 0, 0, 0, -1, 4, -4
250 1
252 FOR R = 1 TO K : FOR C = 1 TO N : READ A(R,C) :
    NEXT C : NEXT R
256 ' Data for objective function CC
258 1
260 DATA 0, 0, 0, 0, 0, 0, 1, 0
262 1
264 FOR C = 1 TO N : READ CC(C) : NEXT C
266 1
268 \text{ V} = 0 : \text{FOR C} = 1 \text{ TO N} : \text{V} = \text{V} + \text{CC(C)} * \text{AO(C)} :
  NEXT C: VNEW = V
270 '
272 ' Main iteration process is the same as in KAREX1.BAS ...
274 1
300 WHILE VNEW/V > T
301 PRINT USING "###"; ITERATION;:
    FOR C=1 TO N:PRINT USING "###.###";XNEW(C);:
    NEXT C: PRINT USING "####.#####"; VNEW/V
302 ITERATION = ITERATION + 1
303 FOR C = 1 TO N : XOLD(C) = XNEW(C) : NEXT C
304 FOR R=1 TO K: FOR C=1 TO N:B(R,C)=A(R,C)*XOLD(C):
    NEXT C: NEXT R
305 FOR C=1 TO N:B(K1,C)=1:NEXT C
306 FOR R=1 TO K1 : FOR C=1 TO K2 : B1(R,C)=0 :
    NEXT C : NEXT R
307 FOR R=1 TO N : FOR C=1 TO K1 : B2(R,C)=0 :
   NEXT C : NEXT R
308 FOR R=1 TO N : FOR C=1 TO N : B3(R,C)=0 :
   NEXT C : NEXT R
309 FOR C=1 TO N : CP(C) = 0 : NEXT C
310 FOR R=1 TO K1: FOR C=1 TO K1:
    FOR I=1 TO N:B1(R,C)=B1(R,C)+B(R,I)*B(C,I):
     NEXT I:
   NEXT C: NEXT R
311 FOR I = 1 TO K1 : B1(I,I+K1)=1 : NEXT I
312 FOR R = 1 TO K1
313
      IF B1(R,R) <> 0 THEN 318
314
        I = R + 1
315
        IF I > K1 THEN PRINT 'Error! BBT is SINGULAR!':
        GOTO 407
```

```
316
        IF B1(I,R) = 0 THEN I = I+1 : GOTO 315
317
        FOR C = 1 TO K2 : SWAP B1(R,C), B1(I,C) : NEXT C
       FOR I = R+1 TO K1:Z = B1(I,R)/B1(R,R):
318
        FOR C=1 TO K2:B1(I,C)=B1(I,C)-Z*B1(R,C):NEXT C:
       NEXT I
319 NEXT R
320 FOR R=K1 TO 2 STEP -1: FOR I = R-1 TO 1 STEP -1: Z = B1(I,R)/B1(R,R):
    FOR C=R TO K2:B1(I,C)=B1(I,C)-Z*B1(R,C):NEXT C:
    NEXT I:NEXT R
321 FOR R=1 TO K1:Z = B1(R,R):
     FOR C=1 TO K2:B1(R,C)=B1(R,C)/Z:NEXT C:
    NEXT R
322 FOR R=1 TO N: FOR C=1 TO K1:
     FOR J=1 TO K1:B2(R,C)=B2(R,C)+B(J,R)*B1(J,C+K1):
      NEXT J:
    NEXT C: NEXT R
323 FOR R=1 TO N: FOR C=1 TO N:
     FOR J=1 TO K1:B3(R,C)=B3(R,C)+B2(R,J)*B(J,C):
      NEXT J:
    NEXT C: NEXT R
324 \text{ FOR R} = 1 \text{ TO N} : B3(R,R) = B3(R,R) - 1 : NEXT R
325 FOR R=1 TO N: FOR C=1 TO N: B3(R,C)=-1*B3(R,C):
    NEXT C: NEXT R
326 FOR R=1 TO N: FOR C=1 TO N:B3(R,C)=B3(R,C)*XOLD(C):
    NEXT C: NEXT R
327 FOR R=1 TO N: FOR C=1 TO N: CP(R)=CP(R)+B3(R,C)*CC(C):
   NEXT C: NEXT R
328 AA=0: FOR C=1 TO N: AA = AA + CP(C)*CP(C): NEXT C
329 AA = SQR(AA) : FOR C=1 TO N : CP(C) = CP(C)/AA :
    NEXT C
330 AA = SQR(N*(N-1))/ALPHA
331 FOR C=1 TO N: XNEW(C) = (AO(C) - CP(C)/AA) \times XOLD(C):
    NEXT C
332 AA=0:FOR C=1 TO N : AA = AA + XNEW(C) : NEXT C
333 FOR C=1 TO N : XNEW(C) = XNEW(C)/AA : NEXT C
334 VNEW=0:FOR C=1 TO N:VNEW=VNEW+CC(C)*XNEW(C):NEXT C
335 WEND
336 1
400 PRINT: PRINT' Tolerance reached: Vnew / Vinitial = ';
     VNEW/V:PRINT
401 PRINT USING "###"; ITERATION; :
    FOR C=1 TO N: PRINT USING "###.###"; XNEW(C); : NEXT C:
    PRINT USING "####.######"; VNEW/V
402 1
403 ' Project solution from simplex back to orthant ...
404 1
405 PRINT: FOR C=1 TO N-2: PRINT XNEW(C)/XNEW(N),:
   NEXT C: PRINT
406 1
407 END
```

```
KAREX1.BAS Accompanies the article "Karmarkar's Algorithm" by Andrew M. Rockett and John C. Stevenson, BYTE, September 1987.
```

```
100 ' -
101 '
102 ' KAREX1. BAS is a Microsoft BASIC Release 5 program that solves EXAMPLE 1 of the article
103 '
104 '
105 '
                  KARMARKAR'S ALGORITHM
106 '
107 '
      by Andrew M. Rockett and John C. Stevenson
108 '
109 'This program was written by Andrew M. Rockett
110 '
111 '
200 '
202 ' N is the number of unknowns and K is the number of equations
204 1
206 N = 3 : K = 1
208 1
210 K1 = K + 1 : K2 = 2*K1
```

```
212 DIM AO(N), XOLD(N), XNEW(N), CC(N), CP(N), A(K,N), B(K1,N), B1(K1,K2), B2(N,K1), B3(N,N)
  214 1
  216 'CC is for the objective function
  218 'B1, B2 and B3 are used for the computation of CP
  220 ' R and C are "row" and "column" indices
  222 1
  224 ' Initially, set XNew = AO, the center of simplex
  226 1
  228 FOR C = 1 TO N: AO(C) = 1/N: XNEW(C) = AO(C): NEXT C
  230 1
  232 ' T is the tolerance
  234 1
 236 T = .001
 238 1
 240 ' ALPHA is usually set equal to 1/4 ...
 242 1
 244 ALPHA = .25
 246 1
 248 ITERATION = 0
 250 1
 252 ' Data for constraint matrix A
 254 1
 256 DATA 2, -3, 1
 258 1
 260 FOR R = 1 TO K: FOR C = 1 TO N: READ A(R,C): NEXT C: NEXT R
 262 1
 264 'Data for objective function CC
 266 1
 268 DATA 3, 3, -1
 270 1
 272 FOR C = 1 TO N : READ CC(C) : NEXT C
 274 1
 276 'Set initial Value to value at center of simplex ...
 278 1
 280 V = 0 : FOR C=1 TO N: V = V + CC(C) *AO(C) : NEXT C: VNEW = V
 282 1
 284 ' Now we can begin the MAIN ITERATION process ...
 286 1
 300 WHILE VNEW / V > T
 301 '
 302 PRINT USING "####"; TERATION; FOR C=1 TO N: PRINT USING "###.####"; XNEW(C); : NEXT C: PRINT USING "####.######"; VNEW/V
 303 1
 304 ITERATION = ITERATION + 1
 305 1
 306 ' Put Xnew into Xold
 307 1
 308 FOR C = 1 TO N : XOLD(C) = XNEW(C) : NEXT C
 309 1
 310 ' Construct the matrix B
 311 '
312 FOR R=1 TO K: FOR C=1 TO N: B(R,C)=A(R,C)*XOLD(C): NEXT C: NEXT R
313 FOR C = 1 TO N | B(K1,C) = 1 | NEXT C
314 1
315 ' Zero matrices to be used in computations . . .
316 '
317 FOR R=1 TO K1 | FOR C=1 TO K2 | B1(R,C)=0 : NEXT C : NEXT R
318 FOR R=1 TO N | FOR C=1 TO K1 | B2(R,C)=0 : NEXT C : NEXT R
319 FOR R=1 TO N + FOR C=1 TO N + B3(R,C)=O: NEXT C: NEXT R
320 FOR C=1 TO N + CP(C) = O + NEXT C
321 '
322 ' Find BBT and put in B1
323 1
324 FOR R = 1 TO K1 | FOR C = 1 TO K1 | FOR I = 1 TO N:B1(R,C)=B1(R,C)+B(R,I)*B(C,I): NEXT I: NEXT C: NEXT R
325 1
326 'Adjoin an identity matrix to BBT
327 1
328 FOR I = 1 TO K1 + B1(I, I+K1)=1 : NEXT I
329 1
330 ' Row reduce BBT I
331 '
332 FOR R = 1 TO K1
333 IF B1(R,R) <> 0 THEN 338
334 I=R+1
335
      IF I > K1 THEN PRINT"Error! BBT is SINGULAR!": GOTO 400
      IF B1(I,R) = 0 THEN I = I+1 : GOTO 335
336
```

```
FOR C = 1 TO K2 : SWAP B1(R,C), B1(I,C) : NEXT C
338 FOR I = R+1 TO K1 : Z = B1(I,R)/B1(R,R): FOR C=1 TO K2:B1(I,C)=B1(I,C)-Z*B1(R,C): NEXT C:NEXT I
339 NEXT R
340 '
341 ' Now back substitute to finish it ...
342 1
343 FOR R = K1 TO 2 STEP -1: FOR I = R-1 TO 1 STEP -1: Z = B1(I,R)/B1(R,R): FOR C = R TO K2:B1(I,C)=B1(I,C)-Z*B1(R,C): NEXT C: NEXT I
   : NEXT R
344 1
345 Remember to make diagonal entries 1's
346 1
347 FOR R=1 TO K1 : Z = B1(R,R) : FOR C = 1 TO K2 : B1(R,C) = B1(R,C)/Z : NEXT C : NEXT R
348 1
349 'BBT Inverse is now in B1 in columns K1+1 to K2
350 '
351 ' Now multiply BBT Inverse by BT and put in B2
352 1
353 FOR R = 1 TO N: FOR C = 1 TO K1: FOR J = 1 TO K1: B2(R,C) = B2(R,C) + B(J,R) *B1(J,C+K1): NEXT J: NEXT C: NEXT R
354 1
355 ' Take THAT and multiply by B and put in B3
356 1
357 FOR R = 1 TO N : FOR C = 1 TO N : FOR J = 1 TO K1:B3(R,C) = B3(R,C) + B2(R,J) *B(J,C) : NEXT J : NEXT C : NEXT R
358 1
359 ' Find I-B3 by subtracting 1's on diagonal and changing signs
361 '
362 FOR R = 1 TO N : B3(R,R) = B3(R,R) - 1 : NEXT R
363 FOR R=1 TO N: FOR C=1 TO N: B3(R,C) = -1*B3(R,C): NEXT C: NEXT R
364 1
365 ' Multiply by D
366 '
367 FOR R=1 TO N: FOR C=1 TO N:B3(R,C)=B3(R,C)*XOLD(C):NEXT C:NEXT R
368 '
369 ' Find projection of CC and call it CP
370 '
371 FOR R=1 TO N: FOR C=1 TO N: CP(R) = CP(R) + B3(R, C) * CC(C): NEXT C: NEXT R
372 '
373 ' Find length of CP and the normalized CP
374 1
375 AA = 0
376 FOR C=1 TO N : AA = AA + CP(C) *CP(C) : NEXT C
377 AA = SQR(AA) : FOR C=1 TO N : CP(C) = CP(C)/AA : NEXT C
378 !
379 'Find a*, project back to get new X ...
380 '
381 AA = SQR(N*(N-1))/ALPHA
382 FOR C=1 TO N: XNEW(C) = (AO(C) - CP(C)/AA) \times XOLD(C): NEXT C
383 '
384 ' And remember to divide by "size" of new X to complete the projective transformation
385 'back to the original simplex
386 '
387 AA = 0
388 FOR C=1 TO N : AA = AA + XNEW(C) : NEXT C
389 FOR C=1 TO N : XNEW(C) = XNEW(C) / AA : NEXT C
390 '
391 ' Find objective function Value at NEW point X
392 1
393 VNEW = 0
394 FOR C=1 TO N : VNEW = VNEW + CC(C) *XNEW(C) : NEXT C
395 1
396 WEND ' End of main iteration loop ...
397 1
398 PRINT: PRINT" Tolerance reached: Vnew / Vinitial = "; VNEW / V: PRINT
399 PRINT USING "####"; ITERATION; : FOR C=1 TO N:PRINT USING "###.###"; XNEW(C); : NEXT C:PRINT USING "####.######"; VNEW/V
400 END
```

BBS'S POSTING BYTENET LISTINGS

Australia:

Grayham Smith
12 Brentwood Road
Flinders Park, South Australia 5025
The Electronic Oracle
300 Baud, CCITT Standard
Telephone: 08-43-3331 Voice
08-260-6686 BBS

Edward A. Romer 31 Warwick Street Killara, Sydney NSW, Australia 2071 *OMEN* 300 & 1200 Baud Telephone: 02-498-2399 Voice (Work) 02-499-2642 Voice (Home) 02-498-2495 BBS

Alan Salmon PCUG Sysop GPO Box 2229 Canberra, A.C.T. 2601, Australia Canberra PC Users Group Inc. 300 & 1200 Baud Telephone: 61-62-58-9967 BBS

Angus S. Bliss
POB 293
Hamilton NSW 2303, Australia
Newcastle Microcomputer Club
300 Baud, CCITT Standard, 8 Bits, 1 Stop, No Parity
Telephone: 049-67-2433 Voice (Angus Bliss)
049-54-9505 Voice (Tony Nicholson)
61-49-685385 BBS

John Hastwell-Batten
POB 242
Dural, NSW 2158, Australia
Tesseract RCPM+
300 Baud, CCITT Standard, 8 Bits, No Parity
Telephone: 02-651-2363 Voice
02-651-1404 BBS

Phil Harding
POB 35
Charnwood A.C.T., Australia 2615
PC-Exchange Bulletin Board
300 & 1200 Baud, CCITT Standard
Telephone: 61-062-581406 Voice
61-62-586352 BBS

Eric Salter
POB 60
Canterbury 3126, Australia
MICOM: The Microcomputer Club of Melbourne
300 Baud
Telephone: 61-3-861-9117 Eric Salter

61-3-762-1386 Peter Jetson (SYSOP) 61-3-762-5088 BBS

Craig Bowen
29 Warrigal Road
Surrey Hills 3127, Vic., Australia
Public Resource #1
300 Baud, CCITT Standard, 8 Bits, 1 Stop, No Parity
Telephone: 03-890-2174

John Blackett-Smith Unit 8 69 Wattle Road Hawthorn 3122, Australia The National Fido Telephone: 613-818-2336

Austria:

Wolfgang Hryzak Bahnstrasse 48 A-2230 Ganserndorf, Austria University of Vienna BBS FIDO 300 Baud, 8 Bits, 1 Stop Bit Telephone: 02282-24094 BBS

Brazil:

Sistema Sampa ATTN: Rizieri Maglio R. Portugal, 202 Jdm Europe - CEP 01446 Sao Paulo - SP - Brazil Sistema Sampa 300 & 1200 Baud, CCITT Standard Telephone: 011-8536273 BBS

Canada:

Leigh Calnek 3036 25th Avenue Regina, Saskatchewan, Canada S4S 1K9 Telephone: 306-586-9253 BBS

Tom Kashuba PCOMM Systems 1411 Fort Street, Suite 2001 Montreal, Quebec, Canada H3H 2N7 Telephone: 514-989-9450 BBS

Gary McCallum
Western Canadian Distribution Center
3420 48th Street
Edmonton, Alberta, Canada T6L 3R5
300 & 1200 & 2400 Baud
Telephone: 403-462-9189 Voice
403-461-9124 BBS

Judson Newell

Canada Remote Systems
Suite 311, 4198 Dundas Street West
Toronto, Ontario, Canada M8X 1Y6
Telephone: 416-231-2383 Voice
416-231-9202 BYTEnet System

Vernon Paige
EPSNLINK
3 McNicoll Avenue
Willowdale, Ontario, Canada M2H 2A6
300 & 1200 Baud
Telephone: 416-494-1380 Voice
416-635-9600 BBS

Terry Smythe
Sysop, Z-Node 40
Muddy Water User Group
55 Rowand Avenue
Winnipeg, Manitoba, Canada R3J 2N6
Telephone: 204-832-3982 Voice
204-945-6713 Voice
204-832-4593 BBS

Chile:

Eurique Benavides Z.
Guillermo Gomara C.
Eliodoro Yanez 2210
Providencia
Santiago de Chile
BIGSA BBS
2400/1200/300 Bell & CCITT
Telephone: 552-749848
10:00 am to 4:00 am (Chilean time)
Downloads & Upiloads
Xmodem/ ASCII

Denmark:

Beverly Kleiman
International Representative
Personal Computer Society of Denmark
Kronprinsensgade 14,
DK-1114 Copenhagen, Denmark
300 Baud, CCITT Standard
Telephone: 01-122518 BBS

England:

Frank Thornley 67 Woodbridge Road, Guildford, Surrey GU21 1JP, United Kingdom CompuLink Telephone: 0-483-65895 Voice

0-483-573337 (300/1200 Baud) BBS 0-483-573338 (1200/2400 Baud) BBS

Finland:

Juha Wiio
Databox Oy
Museokatu 11
00100 Helsinki, Finland
DATABOX FIDO
300 & 1200 Baud
Telephone: 358-0-497904

Vivian Ronald Dwight Suvikuja 3 B 14 02120 Espoo, Finland Micro Maniacs III Fido Node 17 300 & 1200 & 2400 Baud Telephone: 358-0-424524 Voice 358-0-457307 Voice 358-0-467673 BBS

France: Bill Graham.

President
OUF! (Ordinateurs Utilisateurs France)
ATTN: OUFLOG, B.P. 62
10 rue Saint Nicolas
75012 Paris, France
300 Baud, CCITT Standard
Telephone: 331-43-44-06-48 Voice (Bill)
331-43-44-82-65 Voice
331-43-41-61-47 OUFLOG
for BYTE Listings
331-43-40-33-79 OUFTEL
300 & 1200 Baud
331-43-07-95-39 OUFTEL

Dr. Bernard Pidoux
Groupe Des Utilisateurs Francophones D'Informatique
37, Boulevard Saint-Jacques
75014 Paris, France
300 Baud, CCITT Standard
Telephone: 1-47-63-72-50 Voice
1-45-65-10-09 GUFINET
1-45-65-10-11 GUFITEL

Hong Kong:

W. A. Hanafi

SEAnet

Suite 812, Star House,
Tsim Sha Tsui, Kowloon, Hong Kong
ATTN: Christine Wong
Telephone: 5-455088 Voice
5-8937856 SEAnet 1
5-724495 SEAnet 2

Indonesia:

James D. Filgo US Embassy Box R APO SF 96356-5000 Jakarta Computer Society
300 Baud, Bell & CCITT Standard Telephone: 062-21-799-3286 BBS

Israel:

Zohar Levitan POB 10279 Tshala 61102 Israel (Contact for telephone number)

Ireland:

Gerry Clarke 30 Auburn Road Dunlaoire County, Dublin, Ireland Dublin Bay Bulletin Board 300 & 1200 Baud Telephone: 353-01-854179

Italy:

Bruno Bonino MICRO design s.r.l. Via Rostan, 1 16155 Genova, Italy C.B.B.S. CCITT & Bell Standard Telephone: 10-687098 Voice 10-688783 BBS

Giorgio Leo Rutigliano Via degli Oleandri, 7 **POB 175** 85100-Potenza, Italy FIDO-PZ 300 Baud Telephone: 0971-34593 Voice (Work)

0971-54431 Voice (Home) 0971-35447 BBS

Claudio Vandelli Amministratore Unico SOFT SERVICE s.r.l. Via G. B. Morgagni 32 20129 Milano, Italy SOFT SERVICE BBS

300 Baud, CCITT Standard, 8 Bits, No Parity, Full Duplex

Telephone: 02-209231 Voice 02-228467 BBS

Paolo Marraffa Computronix Via De Amicis 76 90145 Palermo, Italy Network Computer Club 300 Baud, CCITT Standard, 8 Bits, 1 Stop Bit, Full Duplex

Telephone: 39-91-266021 BBS 39-91-300229 BBS Japan: Peter Perkins Vice President Honda Trading Company Ltd.

Mail 101 9-91-Chome, Sota Kanda Chiyoda-ku, Tokyo, Japan JANIS 300 & 1200 Baud, CCITT Standard

Telephone: 03-251-0855 BBS

Malaysia:

Ong Boo Huat 3, Jalan Pisang Jalan Kelang Lama, 58000 Kuala Lumpur STARLINK Telephone: 03-7578811 X 116 Voice 03-7576644 BBS

Nigeria:

Chester W. Vlaun MTCE/31 POB 263 Port Harcourt, Nigeria, West Africa 300 Baud Telephone: 234-84-301210 to 301229-3022

Norway:

Robert Hertz Hertz Data Inc. Huitssfeldts Gate 16 N-0253 Oslo, Norway Hacker's Unlimited Telephone: 47-2-431655 Voice 47-2-390521 BBS

Helge Vindenes 5670 FUSA, Norway Costa del Telephone: 47-5-151610 Voice 47-5-234129 BBS

Saudi Arabia:

Larry Layland System Operator DPCS Aramco Box 10063 Dhahran, Saudi Arabia 31311 Dhahran Personal Computing Society Bulletin Board Telephone: 03-873-7851 BBS

Singapore:

Ken Ong 10 Orange Grove Road #04-01 Singapore 1025, Singapore K.B.B.S. 300 & 1200 Baud Telephone: (IDD) 65-734-5825 Voice

(IDD) 65-737-4090 BBS

Sweden:

Jacob Palme Stockholm University Computer Centre-QZ Box 27322 102 54 Stockholm, Sweden BYTECOM Telephone: 46-8-65-45-00 Voice (Work) 08-23-86-60 (300 Baud) 08-23-89-30 (300 Baud) 08-15-59-20 (300 Baud) 08-14-35-00 (1200 Baud) 08-22-81-30 (1200 Baud) 08-24-61-20 (1200 Baud) 08-14-53-70 (1200 Baud)

Carl Nordin Nyakersgatan 8B 531 41 Lidkoping, Sweden A.T.L. 300 & 1200 Baud, CCITT Standard Telephone: 46-510-25280 Voice 46-510-20409 BBS

Switzerland:

Peter M. C. Werner 9, rue de la Colombiere 1260 Nyon, Switzerland OCTET 300 & 1200 & 2400 Baud, CCITT Standard Telephone: 41-22-62-16-54 Voice 41-22-62-18-17 BBS

Albert F. Studer Technical Director Kupfer Electronic AG Soodstrasse 53 Postfach, 8134 Adliswil, Switzerland TRAX 300 Baud, CCITT Standard Telephone: 01-710-81-11 Voice 01-710-44-36 BBS

The Netherlands:

Henk Wevers Cloeckendaal 38 6715 GH Ede, The Netherlands Henk Wevers' Fido Telephone: 31-8380-37156 BBS

West Germany:

Rupert Mohr RMI Nachrichtentechnik GmbH RosstraBe 7 Postfach 1526 D-5100 Aachen, West Germany RMI Net Telephone: 49-241-21145 Voice 45-2410-90528 BYTEnet - DATEX-P 0-26245-2410-90528 User Data - DATEX-P

Rudolf Stricker Unsoeldstr. 20 D-8000 Munich 22, West Germany T-BUS FIDO Telephone: 089-29-38-81 BBS

Announcing BYTE's New Subscriber Benefits Program

Your BYTE subscription brings you a complete diet of the latest in microcomputer technology every 30 days. The kind of broad-based objective coverage you read in every issue. *In addition*, your subscription carries a wealth of other benefits. Check the check list:

DISCOUNTS

- ☑ 13 issues instead of 12 if you send payment with subscription order.
- One-year subscription at \$21 (50% off cover price).
- ▼ Two-year subscription at \$38.
- ▼ Three-year subscription at \$55.
- One-year GROUP subscription for ten or more at \$17.50 each. (Call or write for details.)

SERVICES

- Exchange puts you on-line 24 hours a day with your peers via computer conferencing and electronic mail. All you need to sign up is a microcomputer, a modem, and telecomm software.
- Reader Service: For information on products advertised in BYTE, circle the numbers on the Reader Service card enclosed in each issue that correspond to the numbers for the advertisers you select. Drop it in the mail and we'll get your inquiries to the advertisers.
- TIPS: BYTE's Telephone Inquiry System is available to



subscribers who need *fast* response. After obtaining your Subscriber I.D. Card, dial TIPS and enter your inquiries. You'll save as much as ten days over the response to Reader Service cards.

- ✓ Disks and Downloads:
 Listings of programs that
 accompany BYTE articles are
 now available free on the
 BYTEnet bulletin board, and
 on disk or in quarterly printed
 supplements.
- ✓ Microform: BYTE is available in microform from University Microfilm International in the U.S. and Europe.
- ▼ BYTE's BOMB: BYTE's

 Ongoing Monitor Box is your
 direct line to the editor's desk.

 Each month, you can rate the
 articles via the Reader Service
 card. Your feedback helps us

keep up to date on your information needs.

✓ Customer Service: If you have a problem with, or a question about, your subscription, you may phone us during regular business hours (Eastern time) at our toll-free number: 800-258-5485. You can also use Customer Service to obtain back issues and editorial indexes.

BONUSES

- Annual Separate Issues: In addition to BYTE's 12 monthly issues, subscribers also receive our annual IBM PC issue free of charge, as well as any other annual issues BYTE may produce.
- ☑ BYTE Deck: Subscribers receive five BYTE postcard deck mailings each year—a direct response system for you to obtain information on advertised products through return mail.

To be on the leading edge of microcomputer technology and receive all the aforementioned benefits, make a career decision today. Call toll-free weekdays, 8:30am to 4:30pm Eastern time: 800-258-5485.

And . . . welcome to BYTE country!





Borland's new Turbo C: The most powerful optimizing compiler ever

ur new Turbo C* generates fast, tight, production-quality code at compilation speeds of more than 13,000* lines a minute!

It's the full-featured optimizing compiler everyone has been waiting for.

Switching to Turbo C, or starting with Turbo C, you win both ways

If you're already programming in C, switching to Turbo C will make you feel like you're riding a rocket instead of pedaling a bike.

If you've never programmed in C, starting with Turbo C gives you an instant edge. It's easy to learn, easy to use, and the most efficient C compiler at any price.

What We've All Been Waiting For: a full-featured compiler that produces excellent code in an unbelievable hurry . . . moves into a class all its own among full-featured C compilers . . . Turbo C is indeed for the serious developer . . . One heck of a buy—at any price.

Michael Abrash,

Join more than 100,000 Turbo C enthusiasts. Get your copy of Turbo C today!

Programmer's Journal 35

Technical Specifications

- ☑ Compiler: One-pass optimizing compiler generating linkable object modules. Included is Borland's high-performance Turbo Linker." The object module is compatible with the PC-DOS linker. Supports tiny, small, compact, medium, large, and huge memory model libraries. Can mix models with near and far pointers. Includes floating point emulator (utilizes 8087/80287 if installed).
- Interactive Editor: The system includes a powerful, interactive full-screen text editor. If the compiler detects an error, the editor automatically positions the cursor appropriately in the source code.
- Development Environment: A powerful "Make" is included so that managing Turbo C program development is highly efficient. Also includes pulldown menus and windows.
- Links with relocatable object modules created using Borland's Turbo Prolog* into a single program.
- ☑ Inline assembly code.
- Loop optimizations.
- Register variables.
- MANSI C compatible
- Start-up routine source code included.
 Both command line and integrated environment versions included.
- License to the source code for Runtime Library available.

Sieve benchmark

	Turbo C	Microsoft*
Compile time	2.4	13.51
Compile and link time	4.1	18.13
Execution time	3.95	5.93
Object code size	239	249
Execution size	5748	7136
Price	\$99.95	\$450.00

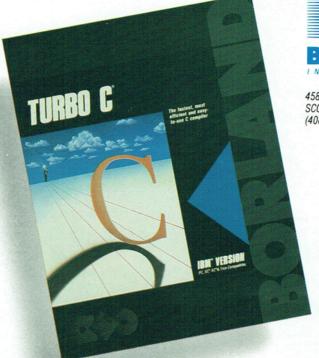
*Benchmark run on an IBM PS/2 Model 60 using Turbo C version 1.0 and the Turbo Linker version 1.0; Microsoft C version 4.0 and the MS overlay linker version 3.51.

Minimum system requirements: IBM PC, XT, AT, PS/2 and true compatibles. PC-DOS (MS-DOS) 2.0 or later. 384K.

For the dealer nearest you or to order by phone call

(800) 255-8008

in CA (800) 742-1133 in Canada (800) 237-1136





4585 SCOTTS VALLEY DRIVE SCOTTS VALLEY, CA 95066 (408) 438-8400 TELEX: 172373

Only \$99.95!